Python 数据抓取技术与实战

潘庆和　赵星驰　编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

数据抓取是获取大数据的一种主要手段。本书主要介绍使用 Python 语言及其相关工具进行数据抓取的方法，通过实例演示在数据抓取过程中常见问题的解决方法。通过本书的学习，读者可以根据需求快速地编写出符合要求的抓取程序。

本书技术性强，注重应用和实战，可供从事数据获取的工程技术人员、理工科院校相关专业的本科生及大数据从业人员使用。

图书在版编目（CIP）数据

Python 数据抓取技术与实战/潘庆和，赵星驰编著．—北京：电子工业出版社，2016.8
ISBN 978-7-121-29884-4

Ⅰ．①P…　Ⅱ．①潘…　②赵…　Ⅲ．①软件工具-程序设计　Ⅳ．①TP311.561

中国版本图书馆 CIP 数据核字（2016）第 217952 号

责任编辑：富　军　　特约编辑：刘汉斌
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：16　字数：410 千字
版　　次：2016 年 8 月第 1 版
印　　次：2016 年 8 月第 1 次印刷
印　　数：3 000 册　　定价：49.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）88254456。

前　言

大数据技术是当前工程和科学技术领域研究的热点。数据科学研究通常包括四个主要环节，即数据获取、数据存储、数据分析及数据可视化。本书主要聚焦数据获取环节。这是其他环节的基础。及时准确地获得丰富详实的数据，可为后续工作奠定坚实的基础，并提高分析结论的可信性和可靠性。

互联网的开放性为数据的获取带来了极大的便利。本书基于 Python 语言的数据抓取技术，主要介绍如何快速准确地从网络上获得所需的数据，构建满足要求的数据集或大数据集。Python 语言是一种通用编程语言，可以应用于各种编程领域，在数据科学领域也是一种十分热门的语言。本书使用 Python 作为数据抓取技术的实现语言，利用 Python 丰富的模块支持和语言特性，解决绝大部分数据抓取中经常会遇到的问题。为了使不了解 Python 语言的读者快速上手，在第 1 章中介绍了阅读本书所需的 Python 语言基础知识。

本书介绍了数据抓取涉及的各类技术问题和解决方法，并按章节进行组织，每章内容基本独立，可使读者在遇到问题时能够快速地进行问题定位。书中的内容侧重于将已有的成熟理论原理和流行框架应用于数据抓取实际问题的解决中。在编写过程中，只侧重介绍应用于数据抓取时的应用方式，并未对某些原理和框架进行详细的描述，感兴趣的读者可以进一步查找相关文献和资料来加深对概念和理论的理解。阅读时，读者可通过运行书中的实例代码，看到现象后再回头去分析，可有助于更好地理解相关的概念和原理，为进一步的研究打下基础。

本书主要面向初学者，读者可基于书中的运行实例进行改造，设计出符合自己要求的数据抓取程序。本书可以迅速用于实战，可供相关专业工程技术人员和高校本科生阅读参考。

感谢首席策划编辑富军老师的辛勤工作！感谢赵星驰老师在外文技术资料方面提供的帮助与协作！

如果读者阅读中发现问题，请及时与我们联系，希望大家多多批评指正。

编著者

目　　录

1 第 1 章 Python 基础

Python 是一种动态类型脚本语言，具有简洁的语法，强大的表达能力，是当前数据科学领域主流的编程语言。其丰富的包和模块的支持可以使 Python 几乎能应用于各个领域。本书使用 Python 作为数据抓取的编程实施语言。

本章主要介绍 Python 编程语言的基础知识及 Python 语言的安装和基本使用方法，通过两个抓取实例讲述 Python 的基本语法及后面各章中将会使用到的语法知识。

在语言版本上选择了 Python3.4 + 作为开发语言，使用 3 以上版本的 Python 在字符串处理上会更加方便。如果没有使用 Python3 以上引入的新特性，那么都可以在对字符串处理并稍加改动后使用 Python2.7 + 执行。随着 Python3 的发展，很多 Python2.7 可使用的库和模块，Python3 都可以使用了，因此建议没有使用过 Python 的读者，可以直接安装 3 以上的版本学习和开发。

在操作系统的选择上，本书的绝大部分实例是在 Ubuntu 14.04 trusty 64 位桌面系统下编写和运行的。如果 Python 版本相同的话，那么在 Windows 下可以直接运行书中的代码。使用 Windows 系统的读者可以利用 Vmware 虚拟机环境在虚拟机中搭建 Ubuntu 系统学习并使用书中的代码，可得到与书中一样的运行效果，也可以直接安装相应版本的 Ubuntu 操作系统。

1.1 Python 安装

Python 是一种跨平台编程语言，可以安装在各类操作系统平台上。在 Ubuntu 14.04 系统上已经安装了 Python，下面介绍在 Windows 环境下安装和使用 Python 的方法。

登录 Python 官网 https://www.python.org/，如图 1-1 所示。在首页单击“Downloads”进入下载页，如图 1-2 所示。

以“Python3.4.3”为例，单击下载。在新打开的下载页面中定位到底部，可以看到该版本的相关下载文件，如图 1-3 所示。

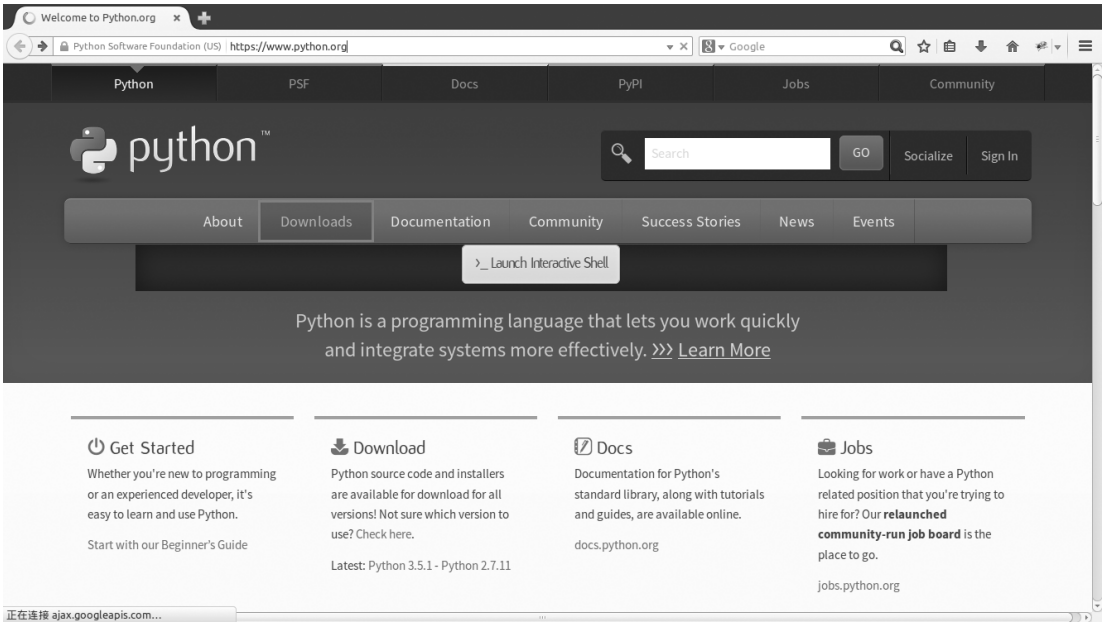


图 1-1　Python 首页

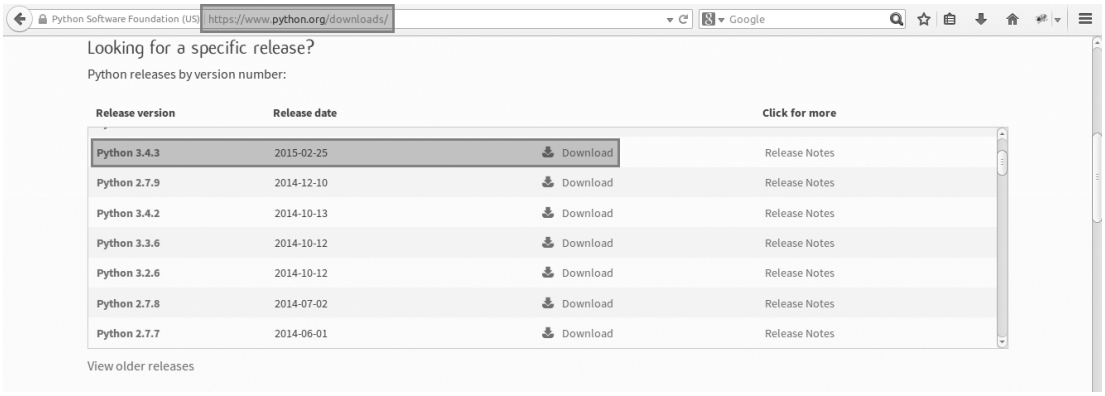


图 1-2　Python 下载页

以 Windows 7 为例，对于 64 位的系统可以选择在如图 1-3 所示中方框的文件下载。下载后双击安装即可，在安装的过程中要注意将 python. exe 添加到环境变量中，这样未来在打开 cmd 窗口时，就可以输入 python 进入 python 的交互环境了。图 1-4 给出了设置的位置，单击“Add python. exe to Path”前面的叉号，在弹出的下拉选项中选择“Will be installed on local hard drive”即可，操作后如图 1-5 所示。接下来就可以进行下一步的安装，直至安装完成。

安装完成后，启动 cmd，在命令行输入“python”，然后回车，即可启动 python 的命令交互环境。启动 cmd 的方式主要有三种。第一种是在左下角单击“开始”按钮，定位到

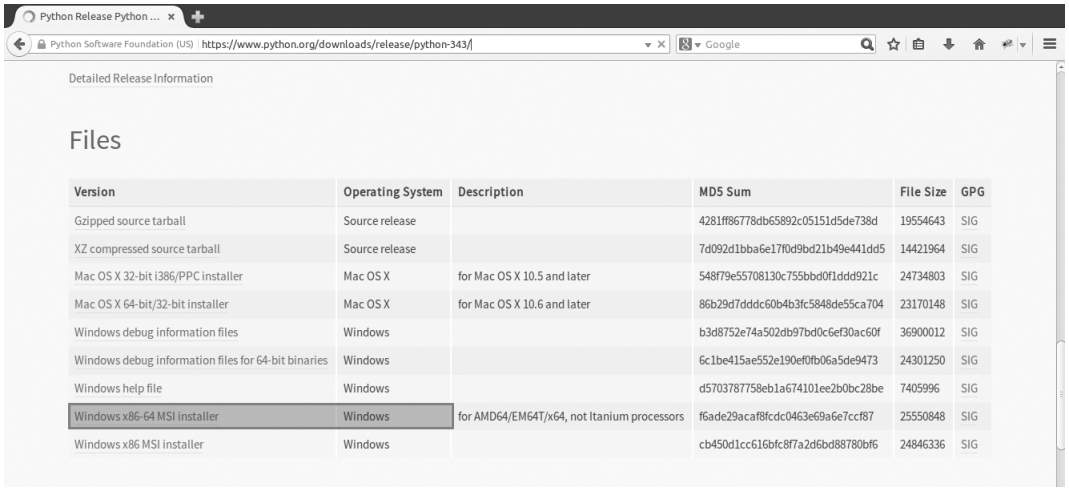


图 1-3　Python 的 Windows 下载文件

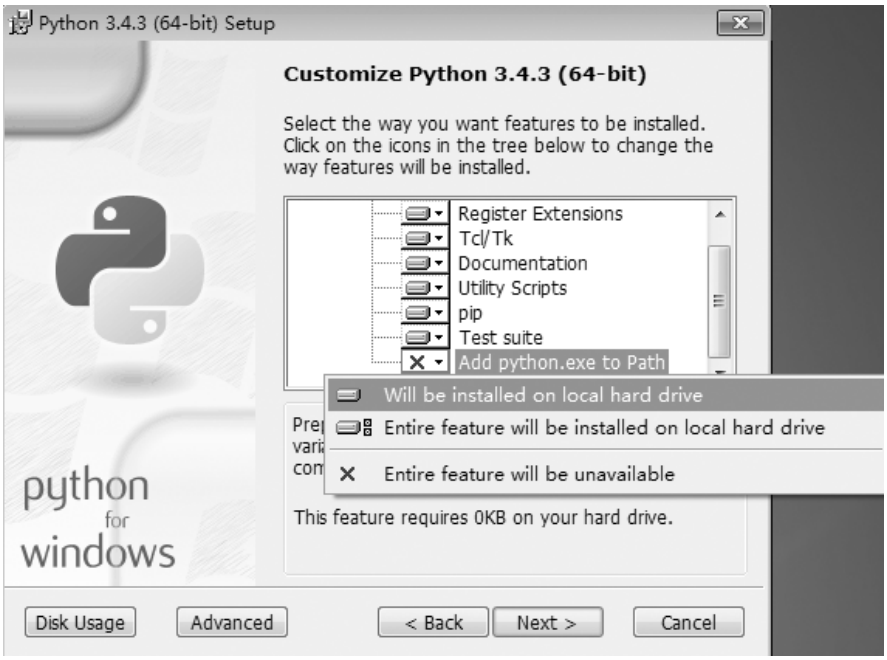


图 1-4　将 python. . exe 添加到系统路径的选项

“搜索程序和文件”的方框，如图 1-6 所示，输入“cmd”后回车，系统会自动搜索出 cmd. exe，如图 1-7 所示。

单击程序下的 cmd. exe，调出命令行，输入 python 执行即可，如图 1-8 所示。

第二种方式是在任何窗口按住 shift 键，单击鼠标右键会弹出菜单，如图 1-9 所示。单击方框内的“在此处打开命令窗口”即可调出命令窗口。

第三种是单击“开始”后，在“附件”中单击“命令提示符”启动命令行，如图 1-10 所示。

图 1-5　将 python. . exe 添加到系统路径

图 1-6　搜索“cmd”的位置

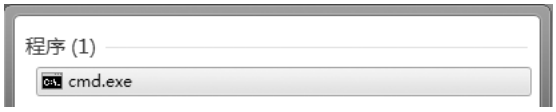


图 1-7　搜索出 cmd. exe

图 1-8　在 cmd. exe 中调出 python 执行环境

图 1-9　右键菜单打开 cmd. exe

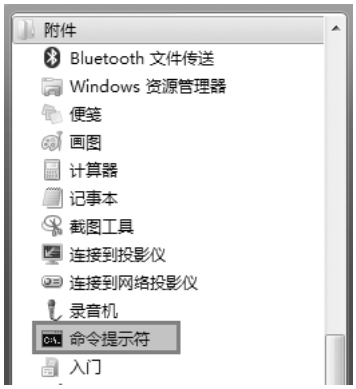


图 1-10　在“附件”中打开 cmd. exe

在 Linux 平台，以 Ubuntu 14.04 为例，在系统安装时就默认安装了 Python 环境，而且在同时默认安装了 Python2.7 和 Python3.4. 的情况下，在 shell 中输入 Python，启动的是 Python2.7，如图 1-11 所示。

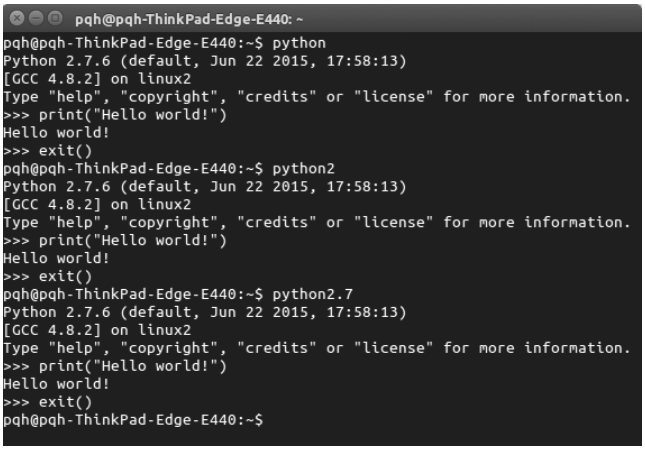


图 1-11　默认启动 Python2.7

如果需要使用 Python3.4，则在 shell 命令行中输入 Python3.4 或 Python3 即可，如图 1-12 所示。

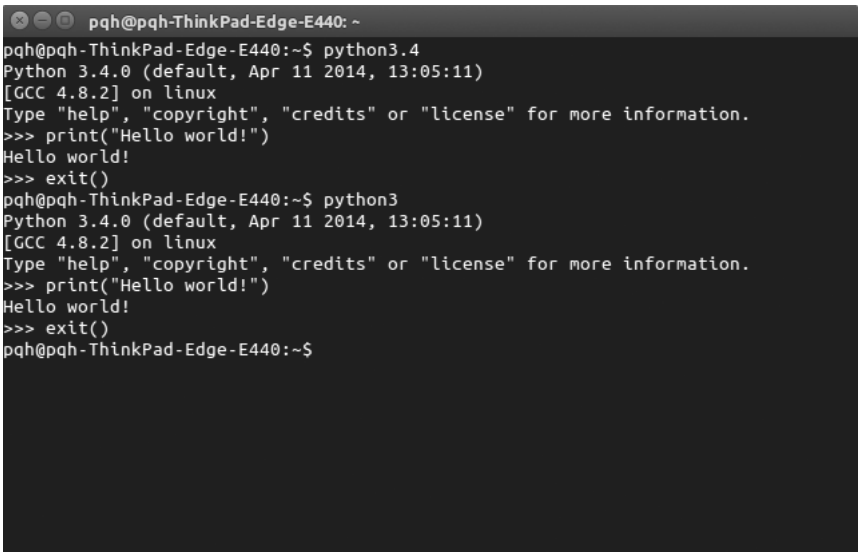


图 1-12　启动 Python3.4

1.2　安装 pip

使用 Python 时，经常需要安装第三方的包文件，可以使用 pip 进行安装，在 Ubuntu14.04下虽然已经安装了 Python2.7 和 Python3.4，但并没有安装相应 Python 版本的 pip 包安装工具，如图 1-13 所示。

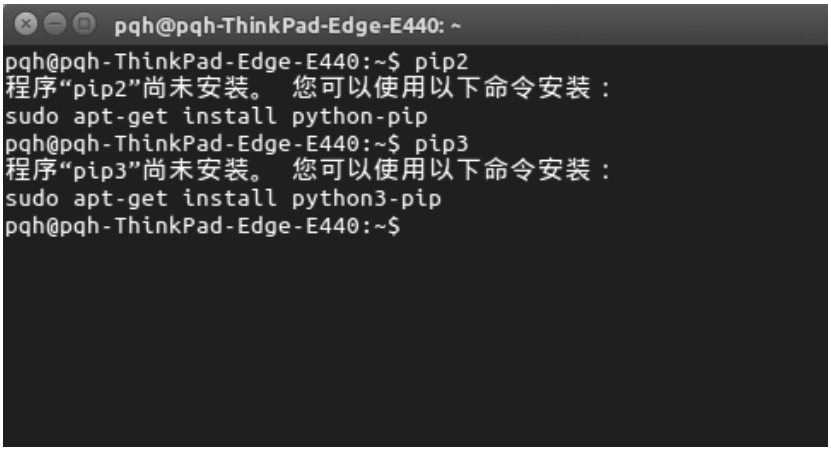


图 1-13　Ubuntu 默认没有安装 pip 工具

系统已经给出了安装的方法，使用提示的命令分别安装即可，命令为

```
sudo apt - get install python - pip
sudo apt - get install python3 - pip
```

安装好后，可以在命令行查看，如图 1-14 所示。

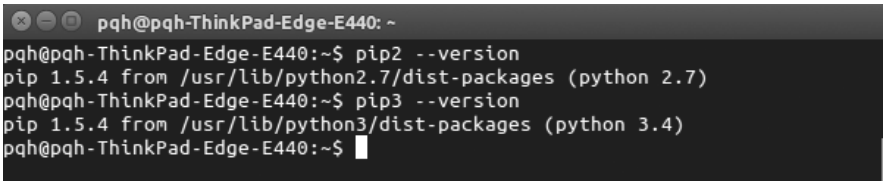


图 1-14　安装了 pip 工具

安装好 pip 后，未来就可以在需要时安装第三方的 Python 包了。

1.3　如何查看帮助

1. help 函数

Python 提供了内置函数 help，可以通过向 help 传递模块、函数、类、方法或 Python 关键字的字符串来查看帮助，如查看有关 os 模块的信息，可在交互环境下输入 help(os)，命令为

```
>>> help( os )
```

在交互环境下会出现相关信息，如图 1-15 所示。

可使用回车或上下箭头按键逐行查看，也可使用翻页键进行翻页查看，在末行“:”后，输入“q”回车，即可退回到交互界面。

当然，也可以在遇到问题时去网络查找，但命令的好处在于可以直接在交互环境中调出帮助文档，可以脱离网络查看，而且文档内容更具针对性。在对 Python 语言及环境有了一定的了解后，通常的查找目标就是函数的返回值和参数信息，帮助文档可以帮助我们快速了解所需信息，提高工作效率。在后面的章节中，当遇到具体的函数时，经常会使用 help 函数帮助我们了解函数的基本用法。

2. dir 函数

dir 也是 Python 的一个内置函数，可以将字符串 dir 作为 help 的参数，查看 dir 函数的

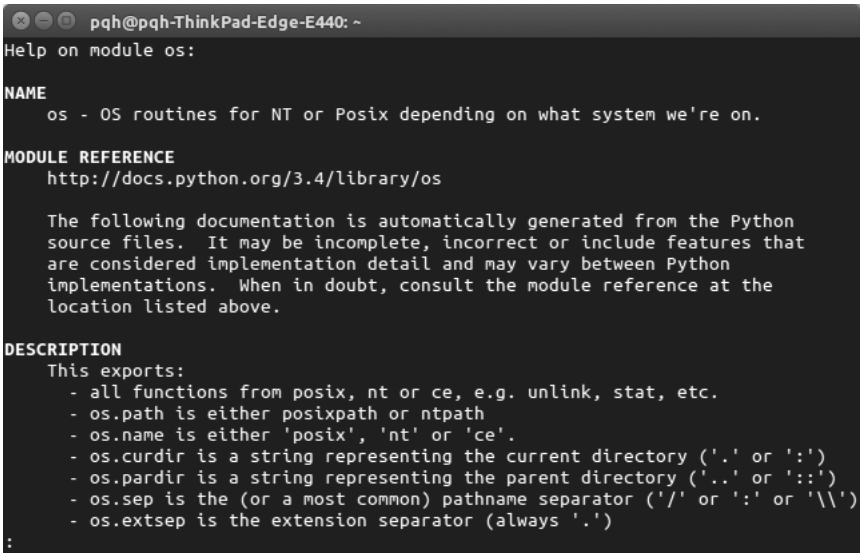


图 1-15　help(os) 显示的内容

使用方法时可输入命令

```
>>> help( dir )
```

显示的内容如图 1-16 所示。

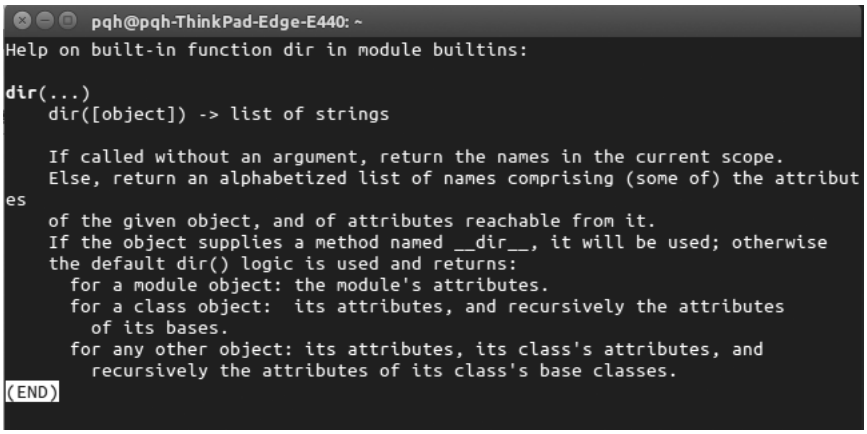


图 1-16　help(dir) 显示的内容

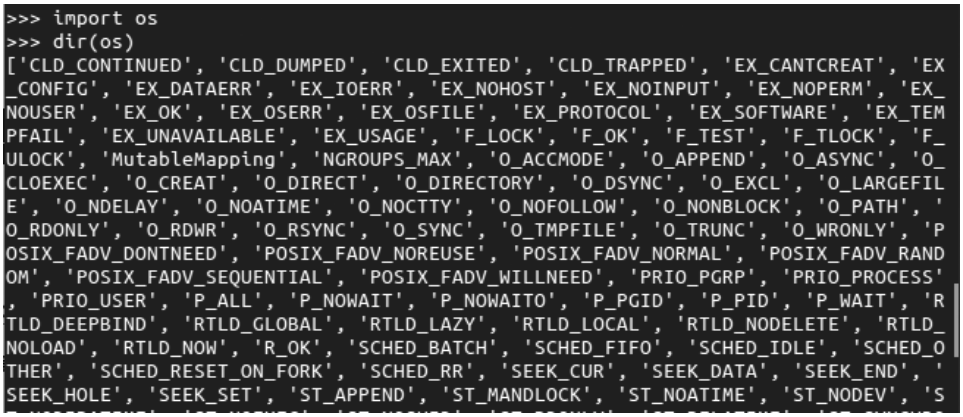
根据上面的描述，可以分析出 dir 函数会返回一个字符串的列表。其中，dir([object]) 表示作为参数的 object 是可选的，即在使用时，既可以传入某个 object 作为参数，也可以不传入任何参数。object 表示对象，在 Python 中的数据类型、模块、函数等都可以视作 object。

如果没有向 dir 函数传递参数，则将返回当前范围的名字列表，如图 1-17 所示。

```
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__']
>>>
```

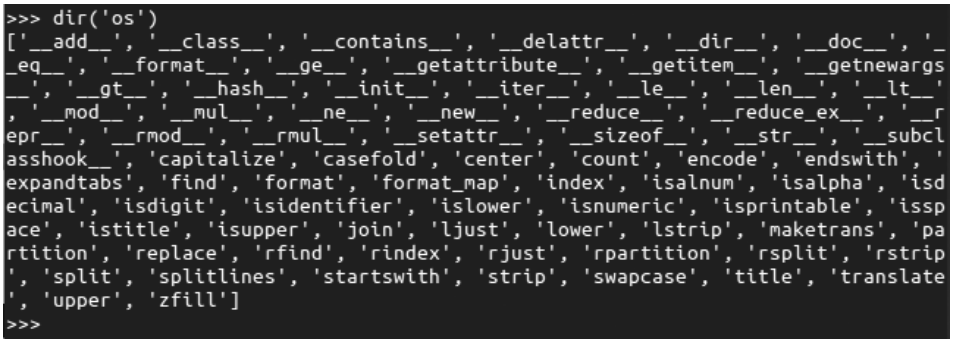
图 1-17　dir( ) 显示的内容

如果将某个 object 作为参数传递给 dir，则 dir 会返回这个对象内所有属性和方法名称的列表。图 1-18 中将 os 模块作为参数传递给 dir，会列出很多 os 模块内属性和方法的名字。如果在使用时对某些 object 的属性和方法名称没有记清的话，则可以通过这种方式查看。注意上面给出的是 dir(os) 的结果。图 1-19 给出的是 dir('os') 显示的内容。



```
>>> import os
>>> dir(os)
['CLD_CONTINUED', 'CLD_DUMPED', 'CLD_EXITED', 'CLD_TRAPPED', 'EX_CANTCREAT', 'EX
_CONFIG', 'EX_DATAERR', 'EX_IOERR', 'EX_NOHOST', 'EX_NOINPUT', 'EX_NOPERM', 'EX_
NOUSER', 'EX_OK', 'EX_OSERR', 'EX_OSFILE', 'EX_PROTOCOL', 'EX_SOFTWARE', 'EX_TEM
PFAIL', 'EX_UNAVAILABLE', 'EX_USAGE', 'F_LOCK', 'F_OK', 'F_TEST', 'F_TLOCK', 'F_
ULOCK', 'MutableMapping', 'NGROUPS_MAX', 'O_ACCMODE', 'O_APPEND', 'O_ASYNC', 'O_
CLOEXEC', 'O_CREAT', 'O_DIRECT', 'O_DIRECTORY', 'O_DSYNC', 'O_EXCL', 'O_LARGEFIL
E', 'O_NDELAY', 'O_NOATIME', 'O_NOCTTY', 'O_NOFOLLOW', 'O_NONBLOCK', 'O_PATH', '
O_RDONLY', 'O_RDWR', 'O_RSYNC', 'O_SYNC', 'O_TMPFILE', 'O_TRUNC', 'O_WRONLY', 'P
OSIX_FADV_DONTNEED', 'POSIX_FADV_NOREUSE', 'POSIX_FADV_NORMAL', 'POSIX_FADV_RAND
OM', 'POSIX_FADV_SEQUENTIAL', 'POSIX_FADV_WILLNEED', 'PRIO_PGRP', 'PRIO_PROCESS'
, 'PRIO_USER', 'P_ALL', 'P_NOWAIT', 'P_NOWAITO', 'P_PGID', 'P_PID', 'P_WAIT', 'R
TLD_DEEPBIND', 'RTLD_GLOBAL', 'RTLD_LAZY', 'RTLD_LOCAL', 'RTLD_NODELETE', 'RTLD_
NOLOAD', 'RTLD_NOW', 'R_OK', 'SCHED_BATCH', 'SCHED_FIFO', 'SCHED_IDLE', 'SCHED_O
THER', 'SCHED_RESET_ON_FORK', 'SCHED_RR', 'SEEK_CUR', 'SEEK_DATA', 'SEEK_END', '
SEEK_HOLE', 'SEEK_SET', 'ST_APPEND', 'ST_MANDLOCK', 'ST_NOATIME', 'ST_NODEV', 'S
```

图 1-18　dir(os) 显示的内容



```
>>> dir('os')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '_
_eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs
__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__'
, '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__r
epr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subcl
asshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', '
expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isd
ecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'issp
ace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'pa
rtition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip
', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate
', 'upper', 'zfill']
>>>
```

图 1-19　dir('os') 显示的内容

这时，dir 的参数是字符串，dir 返回的结果是字符串对象支持的属性和方法名称，与字符串是什么无关，当传入其他字符串时，给出的结果是不变的，都是字符串对象支持的属性和方法名称，如图 1-20 所示。其中的常用方法将在第 2 章专门介绍。

help 和 dir 是两个常用的内置函数，建议读者在遇到问题时综合使用，在后面的实例中，我们通常也会首先调用 help 函数来说明和解释一些概念和内容。

```
>>> dir('os123')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__
eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__
', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '
__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr_
_', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclassho
ok__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expand
tabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal'
, 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'i
stitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition',
 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split'
, 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
 'zfill']
>>>
```

图 1-20　dir(os123) 显示的内容

1.4　第一个实例

下面给出第一个抓取程序。该程序的主要功能是获得电子工业出版社 http://www.phei.com.cn 的首页内容。程序代码包含模块的引入、函数的定义和使用、缩进风格、字符串的使用及简单的分支判断结构等语言内容。下面是程序代码，保存在文件 first_get.py 中。该程序需要 requests 模块，可使用 sudo pip3 install requests 安装。

```
#引入 requests 模块
import requests

#定义 get_content 函数
def get_content(url):
    resp = requests.get(url)
    return resp.text

#定义 url,值为要抓取的目标网站网址
url = "http://www.phei.com.cn"

#调用函数返回值赋值给 content
content = get_content(url)

#打印输出 content 的前 50 个字符
print("前 50 个字符为: ",content[0:50])

#得到 content 的长度
```

```
content_len = len( content)
print( "内容的长度为：",content_len)

#判断内容长度是否大于 40KB
if content_len  >= 40 * 1024：
    print( "内容的长度大于等于 40KB. " )
else：
    print( "内容的长度小于等于 40KB. " )
```

下面对程序进行分析和说明。

1. 注释

在代码中对代码的主要功能和意义均使用注释进行了说明，以“#”开头的行都是注释，如代码的第一行

```
#引入 requests 模块
```

用来说明下一句 import requests 所起的作用。Python 中的注释有两种类型：单行注释和多行注释。以“#”开头的注释属于单行注释，“#”不一定位于行首，也可位于代码行尾，写在代码的后面，如

```
import requests #引入 requests 模块
```

的形式也是可以的。

''' ...''' 和"""..."""可以表示多行注释。前者是用成对出现的三个单引号表示的，后者是用成对出现的三个双引号表示的，如

```
'''
    引入 requests 模块
    引入 requests 模块
    引入 requests 模块
'''
或
"""
    引入 requests 模块
    引入 requests 模块
    引入 requests 模块
"""
```

都是合法的多行注释方法。

2. 模块和包

第一句 import requests 引入 requests 包。

模块（Module）一般是一个 Python 程序文件。程序文件通常以 . py 作为后缀，但模块的命名是不带 . py 后缀的，如文件 a. py，如果将其视作一个模块，那么模块的名称是 a。

包（Package）可用来组织模块并提供一个命名层次，也可以认为包是若干 Python 模块的集合。包的标识是在包文件夹下有一个__ init __ . py 文件，可以使用 import 引入包或模块，此时 Python 创建的都是 module 类型的对象。本书使用各种 Python 自带或第三方提供的包或模块时，并不从引入包或引入模块进行区分，被统称为引入模块。从使用的角度来讲，这不会产生影响，也可以认为 import requests 引入了 requests 模块。

3. 函数和方法

接下来的代码段定义了一个函数，即

```
def get_content(url):
    resp = requests. get(url)
    eturn resp. text
```

函数名为 get_content，接受一个名为 url 的参数，未来在使用时可以传入一个网站的网址作为参数。函数体中只有两行代码。

第一行 resp = requests. get(url)将传入 get_content 函数的参数 url 再传递给 requests 的 get 方法（函数）。requests 支持 get 和 post 方法，分别对应向网站发送请求的 get 方法和 post 方法。这里使用 get 方法向 url 指向的网站发出请求。requests 的 get 方法和 post 方法在发出请求并得到响应后，会返回一个响应对象。本例中的响应对象赋给 resp。

第二行 return resp. text 将获得 resp 对象的 text 属性，并将此属性作为函数的返回值返回，在函数需要返回值时可使用 return 返回。resp 是一个对象，Python 支持面向对象程序设计。text 是 resp 的属性，可以得到响应的文本信息，在 Python 中可以通过“.”得到对象的属性值或调用对象的方法。下面简单总结 Python 中函数的定义方式。

一般可使用 def 定义函数，形式为

```
def 函数名(参数 1,参数 2,参数 3,. . . . . ):
    函数体
    return 返回值
```

函数名应符合 Python 标识符的定义规则；参数可以是各类 Python 允许的数据类型；函

数体是函数执行功能的主体部分；函数可以有返回值，也可以没有，有返回值时，需要使用 return 返回。

方法也是我们常会接触到的概念，一般来讲，在类中定义的函数被称为方法。提到类，就涉及到面向对象的编程机制。下面简单介绍一下 Python 对面向对象程序设计的支持。

使用 class 可以定义一个类，下面的代码定义了一个类，类名为 MyClass。

```
class MyClass:
    def __init__(self,name):
        self.name = name

    def print_name(self):
        print('Hello',self.name)
```

这里需要注意几个问题。第一个是__init__(self,name)。注意，init 的前后各有两个下划线，形式上，__init__是一个函数，出现在类 MyClass 中，可以称为__init__方法。它是一个特殊的方法，可以接受参数来初始化已经实例化的对象。而对象的初始化是由Python面向对象编程时另一个特殊方法__new__来完成的。比如，可以使用下面的语句来完成MyClass的初始化，假设初始化后的对象为 m_class，即

```
m_class = MyClass("Python")
```

执行成功时，m_class 就是 MyClass 初始化后的对象，如果使用__init__和__new__来解释，那么上面的过程相当于下面两句，即

```
m_class = object.__new__(MyClass)
MyClass.__init__(m_class,"Python")
```

一般对于简单问题的处理可以不必显示定义和调用__new__，只需在__init__中给出初始化对象的语句即可，然后使用 m_class = MyClass("Python")方便地进行对象的初始化。

__init__方法有两个参数：self 和 name。其中，self 代表类的实例或称为对象。self 的作用有些类似于 C ++ 中的 this。在类中定义方法时，一般都会使用 self 作为第一个参数，即使这个方法未来在被初始化的对象使用时不需要传入任何参数，如上面定义的 print_name 方法，在定义时需要传入 self，但未来需要调用时无需传入任何参数。在使用 MyClass 类时，首先要进行初始化，利用初始化出的对象来调用定义的各种方法。当调用这些方法时，对象会把自己作为第一个参数传递到相应的方法中，但是调用时不用写出，在对象调用方法时只将定义方法时除去 self 之外的参数依次传入即可。

在__init__方法中只有一句

```
self. name = name
```

该句的作用是在初始化对象时，将传入的参数 name 赋给对象的 name 属性。

在 print_ name 方法中，调用 print 函数将 self. name 输出出来。图 1-21 给出了在命令交互环境下定义和使用 MyClass 的方式。

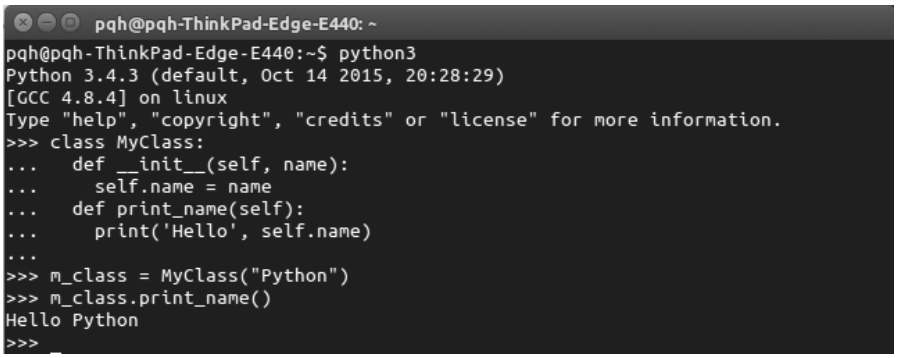


图 1-21　定义和使用 MyClass 的方式

MyClass 被初始化为对象 m_class 时，m_class 属性 name 的值为字符串 Python。在调用 print_name 方法时并没有传递任何参数，但根据前面所述对象，自身已作为第一个参数被默认传入了，以 print_name 方法中的 print 输出时，能够输出对象的 name 属性值 Python。

Python 的面向对象程序设计内容丰富，有很多专门介绍相关内容的书籍。本书只需了解类、对象、方法及方法调用即可。上面的实例虽然简单，但已经涵盖了这几方面的内容。

4. 缩进风格

在 get_content 函数的定义形式中引出了一个 Python 语言十分特别的特点，就是缩进。下面简单介绍一下 Python 的缩进问题。

与大多数编程语言不同，Python 采用缩进的方式表示逻辑层次，而不是用“{ }”括起来的形式。缩进相同的语句块代表处于同一逻辑层次，如上面的 get_content 函数，函数体中的两行代码应处于同一缩进层次，如果在编辑器中输入这两行代码，则应该保证这两句的行首和 def 保持一定的缩进，并且缩进程度相同。

注意，缩进程度相同即可，多缩进一些和少缩进一些是没有差异的。另外，不同的 IDE 在编辑代码的设置上可能有所差异，如在 Pycharm IDE 中可以设置一个 tab 键等于 4 个空格，那么在此 IDE 中使用一个 tab 进行缩进和使用 4 个空格进行缩进，其效果是相同的。但将代码复制到其他 IDE 中时，可能 tab 和空格的数量对应关系不是 1∶4，那么在缩

进上就可能产生问题。最好在编码方式上得到统一，或使用 tab 或使用空格来解决这个问题。

5. 字符串

下面看下一句

```
url = "http://www.phei.com.cn"
```

该句的作用正如注释中描述的，是将待抓取的网址赋值给 url 变量。网址使用双引号括起来，在使用中会被当作字符串处理。单引号也可以表示字符串，如上面的语句写成下面这种形式也是正确的，即

```
url = 'http://www.phei.com.cn'
```

单引号和双引号都可以用来表示字符串。比如，在 shell 中启动 Python3 解释器，输入下面的语句，如图 1-22 所示。

```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> url_1 = "http://www.phei.com.cn"
>>> url_2 = 'http://www.phei.com.cn'
>>> url_1 == url_2
True
>>>
```

图 1-22　单引号和双引号的使用

单引号和双引号都可以表示字符串的一个特点是，单引号中可以包含双引号，双引号中可以包含单引号，这样在处理字符串时会带来一定的方便性，如图 1-23 所示。

```
>>> s1 = "abc"123""
  File "<stdin>", line 1
    s1 = "abc"123""
                ^
SyntaxError: invalid syntax
>>> s1 = "abc\"123\""
>>> s1
'abc"123"'
>>> s2 = 'abc"123"'
>>> s2
'abc"123"'
>>> s1 == s2
True
>>>
```

图 1-23　单引号和双引号的混用

图中希望将字符串 abc“123”赋给变量 s1。如果将 abc“123”使用双引号括起来是不

可以的，因为双引号可以表示字符串，使用双引号括起来之后将是下面这种形式，即

```
"abc"123""
```

是错误的，如图 1-23 所示，解释器会指出出现了语法错误。如果希望将 123 两边的双引号也作为字符串的一部分，则需要进行转义，使用"\"表示不作为限定字符串的标识。图 1-23 给出了这种表示方法。这种方法比较繁琐，特别是需要转义的双引号较多时。如前所述，Python 中可以在单引号中包含双引号，因此可以写成

```
s2 = 'abc"123"'
```

这种形式。这样在表示相同字符串的情况下，写法上会简洁一些。

Python 中的字符串支持索引和切片操作。索引操作可以根据提供的整数索引值在字符串中得到相应的字符值。字符串中第一个字符的索引值为 0，向后依次加 1。假设字符串为 s，则通过 s[0]可获得 s 的第一个字符。这和其他语言中数组的使用方式是一致的。最右端的字符索引为字符串包含字符的个数，即字符串的长度减 1。如果超过这个数值进行索引，则会出现越界错误。字符串的长度可用 Python 内置的 len 函数计算，使用方法为

```
len(字符串)
```

该函数可以返回字符串的长度，那么最右端的字符可以通过 s[len(s)-1]获得。

另外，字符串在进行索引时，Python 支持负数索引，字符串最右端的字符索引值为 -1，右数第二个字符为 -2，依此类推。比如，s[len(s)-1]和 s[-1]都可以获得 s 最右端的字符，可以利用这个特点灵活地使用索引值取得相应的字符，如图 1-24 所示。

图 1-24　Python 字符串索引

切片也和索引一样，是在字符串使用时常用的方式。使用切片可以在字符串中截取一定范围的字符串。假设字符串为 s，则利用切片截取 s 的一段子字符串的形式为

```
s[开始索引:终止索引]
```

开始索引和终止索引 -1 分别指要截取的子字符串左右端字符在原字符串中的索引值。要注意，终止索引 -1 作为最右端字符的索引值是切片方式所规定的。上面给出的索引形式还有一些变化。

（1）只有终止索引，形式为

```
s[:终止索引]
```

这种形式表示从 s 的最左端字符开始截取，直至终止索引 -1 为索引值的最右端字符。

（2）只有开始索引，形式为

```
s[开始索引:]
```

这种形式表示从 s 的开始索引指向的字符开始截取，直至 s 的最后一个字符（包括最后一个字符）。

（3）没有开始索引和终止索引，形式为

```
s[:]
```

这种形式表示将 s 整体返回。

（4）还有一种形式就是按步长切片，这里需要另一个参数表示步长，形式为

```
s[开始索引:终止索引:步长]
```

表示按索引截取时，获得的子字符串中的字符是从开始索引指向的字符开始以步长为间隔进行选取的。Python 字符串切片如图 1-25 所示。

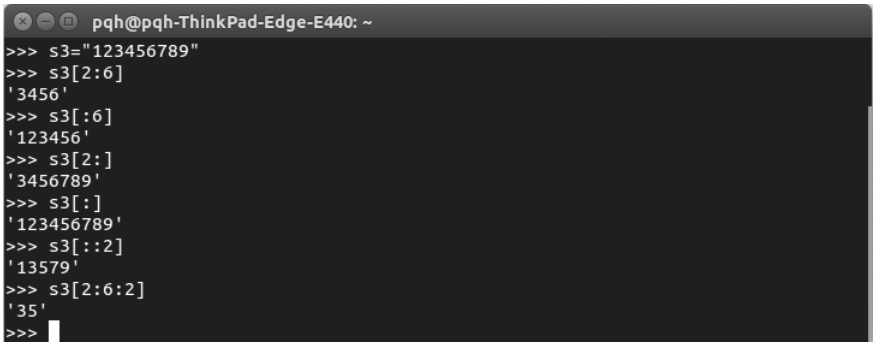


图 1-25　Python 字符串切片

由此看下面的代码就会更加清晰一些。

```
#调用函数返回值赋值给 content
content = get_content( url)

#打印输出 content 的前 50 个字符
print( "前 50 个字符为: ",content[0:50])

#得到 content 的长度
content_len = len( content)
print( "内容的长度为: ",content_len)
```

将定义好的 url 作为参数，调用 get_content 函数，返回的结果赋给 content，根据前面对 get_content 函数的分析可知，content 为 get 请求返回页面的文本，即 HTML 结构的字符串。print 语句用于在控制台输出指定的信息，一般在程序中会使用 print 输出变量的值和其他设定的信息，有助于了解程序的运行情况，在发生问题时方便调试。第一个 print 会输出返回 HTML 文本字符串的前 50 个字符。content 是字符串，利用 len 函数可以获得该字符串的长度。第二个 print 函数输出了获得内容的长度。

Python 自身包含很多字符串处理的方法和模块，同时还有针对特殊文档进行处理的第三方模块，后面在“字符串解析”中会进行更为详细的介绍。下面将介绍字符串运算操作和格式化方面的内容。

字符串支持“+”操作和“*”操作。其中，“+”操作用于连接字符串，“*”操作用于重复字符串，如图 1-26 所示。

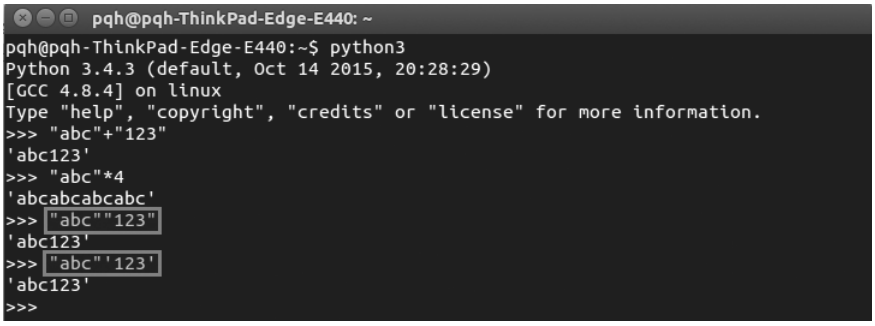


图 1-26　Python 字符串操作

图中，前两个实例给出了“+”和“*”运算，后两个实例没有使用“+”运算，而是直接将两个字符串写在了一起，这也可以完成两个或多个字符串的连接。

对于格式化，在字符串的各种语言中都有相应的函数，Python 支持使用% 和 format 函数进行字符串的格式化。下面分别介绍这两种方式，特别是 format，在后面的内容中会经常用到。

在前面的实例中使用了 print 语句进行输出，如

```
print("前 50 个字符为：",content[0:50])
```

print 函数包含两个字符串参数，输出的内容相当于将两个参数连接在一起，适用于简单的情况。有时输出的内容较多，并且希望以一定的格式将输出的内容拼接在一起时，这种简单的输出形式就无法做到了，需要使用格式化操作符，即%，如图 1-27 所示。

```
>>> print("内容是%s" % "abc")
内容是abc
>>> print("内容是%s,长度是%d" % ("abc", len("abc")))
内容是abc,长度是3
```

图 1-27　使用% 格式化操作符

格式化操作符% s 和% d 分别代表字符串和整数，表示未来在相应的位置将接受字符串或整数。出现% s 和% d 的字符串被认为是模板字符串，而% 后连接的元组被认为是将要向模板中提供的具体内容值。如果只有一个元素被放入模板，也可以不写成元组形式，直接提供参数即可，如 print("内容是% s" % "abc")。元组将在后面介绍，这里只需了解元组类似于其他语言中的数组，元素从 0 开始索引，元组是不可变数据类型，只能读取其中的元素，而不能对元素进行修改。% 后连接的元组中的元素会依次填入前面字符串模板中相应的% s 或% d 等格式化操作符的位置。如果% 后连接的元组中元素的个数和模板中格式化操作符的个数不一致，或者元组中元素类型和对应格式化操作符所表示的类型不一致都会产生错误。Python 有很多格式化操作符用于不同的格式化场合，具体使用时可参考 Python 文档。

另一个重要的字符串格式化方法是使用 format 函数。该函数由字符串调用。与% 格式化操作符相比，使用 format 函数时字符串中不使用% s 或% d 等格式化操作符，而是使用“{}”取代格式化操作符作为占位的标识。在进行格式化时，“{}”将被实际的内容所取代，“{}”中可以填入内容指导 format 进行格式化，如图 1-28 所示。

```
>>> print("内容是{},长度是{}".format("abc", len("abc")))
内容是abc,长度是3
>>> print("内容是{0},长度是{1}".format("abc", len("abc")))
内容是abc,长度是3
>>> print("内容是{1},长度是{0}".format("abc", len("abc")))
内容是3,长度是abc
>>> print("内容是{1},长度是{1}".format("abc", len("abc")))
内容是3,长度是3
>>> print("内容是{c},长度是{l}".format(c="abc", l=len("abc")))
内容是abc,长度是3
```

图 1-28　使用 format 格式化字符串

图中所示第一个实例中的{}内不包含任何内容，在进行格式化时会被 format 中的参数依次取代。后面三个实例中的{}中填入了数字，可以将 format 的参数作为元组考虑，那么

{}内的数字就是元组内元素的索引，如

```
print("内容是{0},长度是{1}".format("abc",len("abc")))
```

{0}将被元组的第 0 个元素，即参数"abc"取代，{1}将被元组的第 1 个元素，即参数 len("abc")取代。下面的两个实例将{0}和{1}的不同占位情况列举出来，这样的格式化可能没有意义，主要是演示{}内加入整数索引和 format 参数间的联系。

最后一句

```
print("内容是{c},长度是{l}".format(c="abc",l=len("abc")))
```

{}里面填入的不再是数字，而是一个字符串，要求 format 的参数也要有相应的变化，在参数中，c="abc",l=len("abc")，那么{c}和{l}就会被"abc"和 len("abc")替代。format 中参数的顺序没有关系，不会对替代造成影响。上句的输出和下面这句是一样的，即

```
print("内容是{c},长度是{l}".format(l=len("abc"),c="abc"))。
```

6. 控制结构

接下来的代码是一段控制结构，即

```
#判断内容长度是否大于40KB
if content_len >= 40*1024:
    print("内容的长度大于等于40KB.")
else:
    print("内容的长度小于等于40KB.")
```

该段代码的主要作用是判断返回的内容长度是否大于等于 40KB。如果是，则输出内容长度大于等于 40KB，否则输出内容长度小于等于 40KB。在 if 后的表达式为

```
content_len >= 40*1024
```

它的真假值将决定控制分支的走向。真假值使用布尔（bool）类型描述，真用 True 表示，假用 False 表示。若 content_len >= 40*1024，则表达式为 True，否则为 False。

某些数据类型的特殊值在进行逻辑判断时也具有布尔值作用。False、None、[]（空列表）、{}（空字典）、""、set()（空集合）、0、0.0 等出现在需要布尔值的位置时被认为是假。其中，列表、字典和集合等数据结构将在后文继续介绍。图 1-29 的代码中给出空列表的实例。对于一个空列表 l=[]，在 if 进行分支判断时被认为是 False。

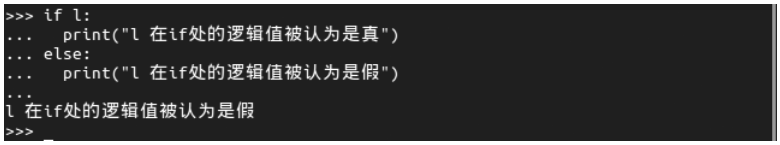


图 1-29　空列表在 if 中被认为是 False

l 在 if 处的逻辑值被认为是假。

控制结构是在编程时常用的结构，常与循环结构结合使用，用来描述和处理复杂的逻辑业务流程。循环结构将在后文讲解。下面将对 Python 中的控制结构进行简单介绍。Python中可使用下面形式的 if. . . elif. . . elif. . . else 结构实现控制流，对程序的运行分支进行控制。

```
if 表达式 1：
    语句 1
elif 表达式 2：
    语句 2
elif 表达式 3：
    语句 3
...
else：
    语句 n
```

各分支依据表达式的逻辑值进行分支控制，如果表达式的逻辑值为 True，则执行相应的分支，否则进入后续的分支表达式判断，继续重复这一过程。在实际应用中，elif 部分可以不出现，同时 else 部分也是可选的。像上面这种完整的形式，与其他语言中的 switch. . . case 用法和作用类似。图 1-30 给出了基本的使用方法。

图 1-30　if. . . elif. . . else 基本的使用方法

至此，通过第一个抓取程序，介绍了一些 Python 中常用的语法结构和数据类型的使用

方法。在一个实例中很难包含和体现 Python 全部的语言特性，后面还会通过实例介绍其他一些常用的语法知识。这些知识都是后面实际应用中经常用到的。前面已经将这段代码保存在 first_get. py 中，图 1-31 为在 shell 中执行 Python 程序。

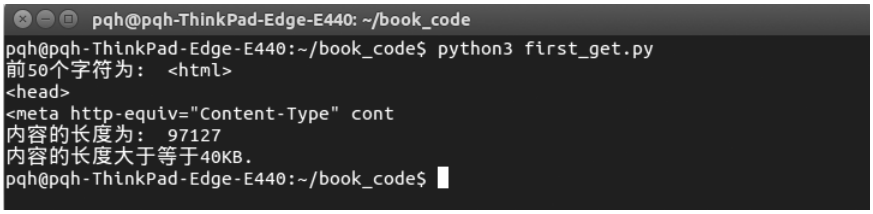


图 1-31　在 shell 中执行 Python 程序

7. if __name__ ==' __main_' :

在前面的代码中，我们定义了函数 get_content，定义了需要使用的字符串 url，对函数进行调用后，通过 print 输出相应的信息，并且结合分支控制结构对获得内容的大小信息做了进一步的判断和输出，所有的这些内容写在了一个文件中（通常称为脚本文件）。在该文件中，函数及变量的定义和使用混合在一起，写起来灵活、方便。有时，我们希望在解决问题时将总结和抽象出的方法放在一起形成一个模块或包，供自己和其他用户方便地使用，就像前面代码中引入 requests 一样。比如，希望将前面的代码文件 first_get. py 作为一个模块在 snd_get. py 中引入，使在 snd_get. py 中的代码能够使用first_get. py 中的 get_content 函数，那么 snd_get. py 文件可能需要这样写，将下面的代码保存在 snd_get. py 文件中，并与 first_get. py 保存在同一个文件夹中。

```
import first_get

url = 'http://www.sina.com.cn'

sina_content = first_get.get_content(url)

print('----------------------------------')
print('snd_get.py 得到内容前 50 个字符:',sina_content[0:50])
print('snd_get.py 得到内容长度:',len(sina_content))
```

在该段代码中引入 first_get 模块，并定义自己要抓取的网址。利用 first_get 模块中的 get_content 函数完成抓取，将抓取到的文本保存在 sina_content 中，然后与前例一样输出抓取内容的前 50 个字符，并输出抓取内容的长度。执行 snd_get. py 如图 1-32 所示。

可见，代码能够正常运行并得到结果，在虚线下的内容就是 snd_get. py 输出的内容。

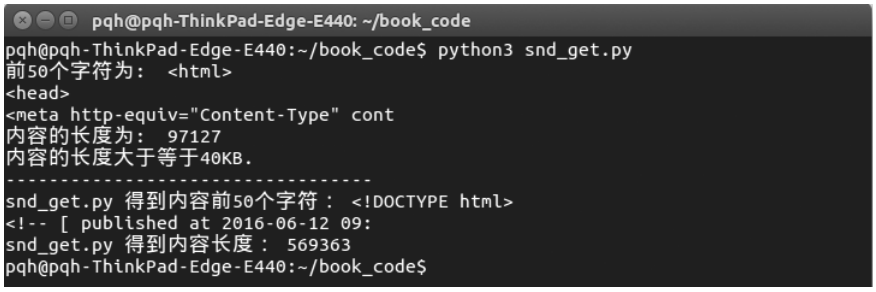


图 1-32　执行 snd_get. py

但同时也可以看到，first_get. py 中的内容也被执行了。这是因为在 snd_get. py 执行前，会根据 import 将 first_get. py 中的代码加载进来，因此在执行时会将这部分代码也执行一遍，导致图 1-32 出现的结果。为了避免出现这种情况，可以在 first_get. py 中使用

```
if __name__ ==' __main__' :
```

语法。其中，__name__是 Python 自身维护的一个内建变量，当一个 . py 文件被执行时，__name__的值为' __main__' ；当被其他文件利用 import 引入时，__name__的值为模块的名字。下面通过实例看一下含义。首先给出 name_test_1. py 文件，内容为

```
print(__name__)
```

给出 name_test_2. py 文件，内容为

```
import name_test_1
print(__name__)
```

代码很简单，主要是演示__name__在不同情况下的不同值。将两个文件保存在一个文件夹中，在 shell 环境下分别运行两段代码，如图 1-33 所示。

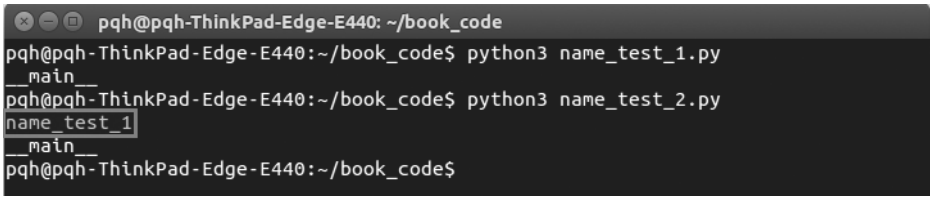


图 1-33　__name__的取值情况

如前所述，当一个 . py 文件被执行时，__name__的值为' __main__' ，因此使用 python3执行文件时，输出' __main__' ；在 name_test_2. py 中，将 name_test_1. py 文件作为模块引入，此时 name_test_1. py 不是被执行，而是被作为模块引入的状态，并且根据刚才

snd_get. py的实例，被引入模块中的代码也会被执行。因此，name_test_1. py 中的 print 语句会被执行，因为被引入，所以输出为模块的名字，如图中方框所示。

根据对__name__的了解，可以在程序中利用其值变化的特点，结合

```
if __name__ ==' __main__' :
```

这种分支结构，使代码文件既可被单独执行，也可被其他模块引入，只调用相应的方法函数，而不重复执行代码。其具体做法是将代码中的关键定义放在该分支结构之外，将代码执行的主要功能放在这个分支结构中，可修改 first_get. py 的内容，如

```
#引入 requests 模块
import requests

#定义 get_content 函数
def get_content(url):
    resp = requests. get(url)
    return resp. text

if __name__ ==' __main__' :

    #定义 url,值为要抓取的目标网站网址
    url = "http://www. phei. com. cn"

    #调用函数返回值赋值给 content
    content = get_content(url)

    #打印输出 content 的前 50 个字符
    print("前 50 个字符为:",
          content[0:50])

    #得到 content 的长度
    content_len = len(content)
    print("内容的长度为:",content_len)

    #判断内容长度是否大于 40KB
    if content_len >= 40 * 1024:
        print("内容的长度大于等于 40KB. ")
```

```
    else:
        print("内容的长度小于等于 40KB.")
```

保持 snd_get.py 的内容不变，分别执行两段代码的结果如图 1-34 所示。

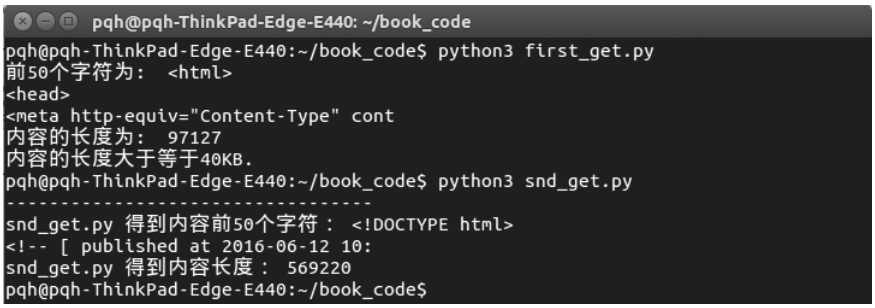


图 1-34　分别执行两段代码的结果

通过执行结果可以看出，first_get.py 在单独执行时，功能没有受到影响，并且在 snd_get.py中引入 first_get 模块。其中，功能代码并没有执行，而只引入了所需的 get_content 函数。

1.5　文件操作

在前面的实例中将抓取到的 html 字符串内容保存到字符串 content 中，使用 print 输出 content 的前 50 个字符和内容的长度。有时希望将抓取到的内容以文件的形式保存下来并在需要的时候读取。Python 提供了文件操作的相关函数，可以很方便地对文件进行读/写操作。在下面的代码中给出了文件操作的两种方式：一种是常规的打开文件、操作文件、关闭文件的方式；另一种是使用 with 的形式。两种方式完成的功能是相同的，都是将 content 的内容先保存在文件中，再从保存的文件中读取出来，输出读取内容的前 50 个字符。下面的代码在 first_get.py 的基础上增加了上述内容，保存在文件 first_get_and_save_file.py 中，即

```
#引入 requests 模块
import requests


#定义 get_content 函数
def get_content(url):
    resp = requests.get(url)
```

```
    return resp.text

if __name__ == '__main__':

    #定义 url,值为要抓取的目标网站网址
    url = "http://www.phei.com.cn"

    #调用函数返回值赋值给 content
    content = get_content(url)

    #打印输出 content 的前 50 个字符
    print("前 50 个字符为:",content[0:50])

    #得到 content 的长度
    content_len = len(content)
    print("内容的长度为:",content_len)

    #判断内容长度是否大于 40KB
    if content_len >= 40*1024:
        print("内容的长度大于等于 40KB.")
    else:
        print("内容的长度小于等于 40KB.")

    # 方式 1
    #文件的写入
    print('-'*20)
    print('方式 1:',文件写入)
    f1 = open('home_page.html','w',encoding='utf8')
    f1.write(content)
    f1.close()

    #文件的读取
    print('方式 1:',文件读取)
    f2 = open('home_page.html','r',encoding='utf8')
    content_read = f2.read()
    print("方式 1 读取的前 50 个字符为:",content_read[0:50])
    f2.close()
```

```
# 方式 2
# 文件的写入
print( '-' * 20)
print( 方式 2:', 文件写入 )
with open( home_page_2. html', 'w' ,encoding = utf8' ) as f3:
    f3. write( content)
# 文件的读取
print( 方式 2:', 文件读取 )
with open( home_page_2. html', 'r' ,encoding = utf8' ) as f4:
    content_read_2 = f4. read( )
    print( "方式 2 读取的前 50 个字符为: " ,content_read_2[0:50])
```

方式 1 采用了传统的文件操作方式。Python 提供了 open 函数进行文件打开操作，使用时需要传入参数。常用的参数有文件名、打开模式、编码方式，即

```
f1 = open( home_page. html', 'w' ,encoding = utf8' )
```

在该句中，文件名为 home_page. html，打开模式是 w，代表 write，表明打开文件将要执行文件的写入操作。在该方式下，如果指定的文件名对应的文件不存在，则会建立一个指定名称的新的空文件，然后写入；如果指定的文件名对应的文件存在，则对应文件的内容会被重新写入，也就是说，以前的内容会被覆盖掉。如果希望将要写入的内容添加到现有文件内容的末尾而不是覆盖，则可以使用模式 a，代表 append，追加的意思。参数 encoding 表示写入文件时使用的编码，utf8 是一种常用的编码形式，其他常用的编码还有 gbk、gb2312 等。编码是在数据抓取中常会遇到的问题。

open 函数返回的是文件流对象，上面的这句代码将该对象保存在 f1 变量中，接下来就可以使用文件对象提供的方法对文件进行操作了。本实例中使用了 write 和 read 方法。两种方法分别实现了文件的写入和读取操作。write 方法接受一个字符串参数，并将字符串写入到文件中。read 方法可接受一个表示读取字节数量的整数作为参数，然后只读取相应数量的字节。本实例中没有给 read 方法传递参数，表示读取全部的文件内容。采用 read 方法读取后，将返回一个包含读取内容的字符串，代码将该字符串保存在 content_read 中，这样就可利用各种对字符串操作的方法根据需要对 content_read 进行操作了。文件对象还包含 readline 和 readlines 两种方法，可以实现对文件的按行读取。文件对象使用完毕后需要关闭。通过调用 close 方法可以关闭。

方式 2 使用了 with 语句。with 语句在使用前需要设置，使用后，需要清理的使用场景提供了一种简洁的使用模式。方式 2 中的文件操作方式是一种常规的方式，包括文件的打开、文件的操作、文件的关闭，可以使用 with 简洁地表示，如上面代码使用 with 后无需关

闭文件，with 会帮我们进行清理。with 语句不仅可以用于文件操作的场景，在很多其他情况下也都可以使用 with。

运行上面代码的结果如图 1-35 所示。

图 1-35　first_get_and_save_file. py 的执行结果

Python 语言简洁强大，语法灵活。由于篇幅的限制，本章只介绍后面在抓取中涉及的一些语法知识和数据结构。前面的第一个抓取程序 first_get. py 已经介绍了 Python 程序的一些基础语法知识。下面将介绍 Python 其他的一些常用语法和数据结构，先通过简单的实例并针对每个语法知识和数据结构进行介绍，然后通过一个实例将这些内容综合起来。

1.6　循环

Python 支持 while 循环和 for 循环。while 循环根据表达式的逻辑真假值来决定是否执行循环体。for 循环主要是对一个序列进行遍历。比如，下面的代码将 x 的初始值设为 0，while 循环在执行时判断 x 是否小于 5，如果小于 5，则输出“x 数值 <5”，并将 x 加 1，然后 while 继续根据新的 x 值判断 x <5 是否成立。若成立，则继续重复上面的动作，否则退出，如图 1-36 所示。

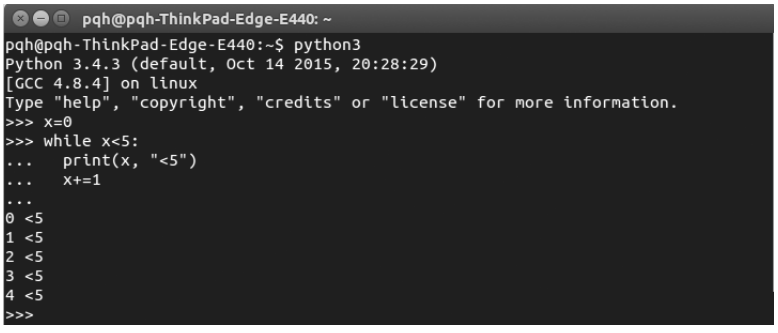


图 1-36　while 循环

for 循环也是一种常用的循环形式，下面的代码使用 for 循环来实现上面相同的功能，如图 1-37 所示。

```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> for x in range(0, 5):
...     print(x, "<5")
...
0 <5
1 <5
2 <5
3 <5
4 <5
>>>
```

图 1-37　for 循环

可根据需要选择 while 循环和 for 循环。循环可以嵌套，就是常说的多重循环，如图 1-38 所示。

```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> for i in range(0, 3):
...     for j in range(0, 2):
...         print("i=", i, "j=", j)
...
i= 0 j= 0
i= 0 j= 1
i= 1 j= 0
i= 1 j= 1
i= 2 j= 0
i= 2 j= 1
>>>
```

图 1-38　循环嵌套

上面给出了二重循环的实例。下面给出循环和分支循环嵌套的实例，如图 1-39 所示。

```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> for i in range(0, 5):
...     if i%2 == 0:
...         print(i,"is even.")
...     else:
...         print(i,"is odd.")
...
0 is even.
1 is odd.
2 is even.
3 is odd.
4 is even.
>>>
```

图 1-39　循环和分支循环嵌套

1.7 异常

Python 支持异常处理机制，在程序出现错误和执行问题时，可以利用 try... except... 进行异常捕获和处理，如图 1-40 所示。

```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> try:
...   1/0
... except Exception as e:
...   print(e)
...
division by zero
>>>
```

图 1-40　try... except... 异常捕获和处理

Python 的异常处理机制十分灵活，结构支持很多的变化，具体用法可参考官方教程手册。本书中只使用图中的这种基本结构。

1.8 元组

元组和列表是 Python 常用的数据结构，类似于其他语言中的数组类型。元组和列表与前面介绍的字符串一样，都支持索引和切片操作，可以方便地保存和读取数据序列。相比于元组，列表是一种更为常用的数据结构。下面先介绍元组，后面将介绍列表，读者可以对比学习，加深理解。

元组也是一种序列类型，可以像列表一样使用元素的索引值进行索引，定义元组时可以使用“,”或使用“()”，如图 1-41 所示。

```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> t1=1,2,3
>>> t1
(1, 2, 3)
>>> t2=(1,2,3)
>>> t2
(1, 2, 3)
>>> t1==t2
True
>>>
```

图 1-41　元组的定义

其中，t_1 = 1、2、3 被称为元组的打包（packing），与之相反的过程被称为解包（unpacking），如图 1-42 所示。

```
>>> a,b,c=t1
>>> a
1
>>> b
2
>>> c
3
>>>
```

图 1-42　元组 unpacking

在解包时，要求等号左边的变量个数和等号右边元组的元素个数相同。Python 的多元赋值就是打包和解包两个过程的组合，如图 1-43 所示。

```
>>> x,y,z=100,200,300
>>> x
100
>>> y
200
>>> z
300
>>>
```

图 1-43　Python 多元赋值

元组是不可变的，列表是可变的，意味着可以通过元素索引值访问元组中的元素，但不可以修改，如图 1-44 所示。

```
>>> t1[0]
1
>>> t1[0]=4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

图 1-44　元组的索引

当某一元素是可变类型时，可以通过修改该可变类型以达到修改元素的目的，如图 1-45 所示。

```
>>> t3=(1,2,[3,4])
>>> t3[2][1]=5
>>> t3
(1, 2, [3, 5])
>>> t3[2]=[3,5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

图 1-45　可变类型元素的修改

为了保持元组不可变的特性，需要保证每个元素都具有不可变的特性。

可以使用循环对元组进行遍历，如图 1-46 所示。

```
>>> for elem in t1:
...     print(elem)
...
1
2
3
>>>
```

图 1-46　元组元素的遍历

元组也支持切片操作，如图 1-47 所示。

```
>>> t1[:]
(1, 2, 3)
>>> t1[1:]
(2, 3)
>>> t1[1:3]
(2, 3)
>>>
```

图 1-47　元组的切片操作

1.9　列表

列表是一种常用的数据结构，与元组相比，列表的元素可变。Python 提供了很多方法用来对列表进行操作。

（1）可以使用“[ ]”来定义一个列表。列表元素的索引从 0 开始，越界将出现错误，len 可以获得列表长度，如图 1-48 所示。

```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> l1=[1, 2, 3, 'abc']
>>> l1
[1, 2, 3, 'abc']
>>> l1[0]
1
>>> l1[3]
'abc'
>>> l1[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> len(l1)
4
>>>
```

图 1-48　列表的定义、索引和长度获取

（2）列表支持切片和索引操作，如图 1-49 所示。

（3）列表元素是可变的，可以利用索引来修改元素，如图 1-50 所示。

图 1-49　列表的切片和索引操作

图 1-50　列表元素的修改

（4）Python 提供了 append 和 extend 方法对列表进行修改。其中，append 可以向列表末尾追加新的元素，entend 可以向列表末尾追加另一个序列（如元组或列表）中的元素，如图 1-51 所示。

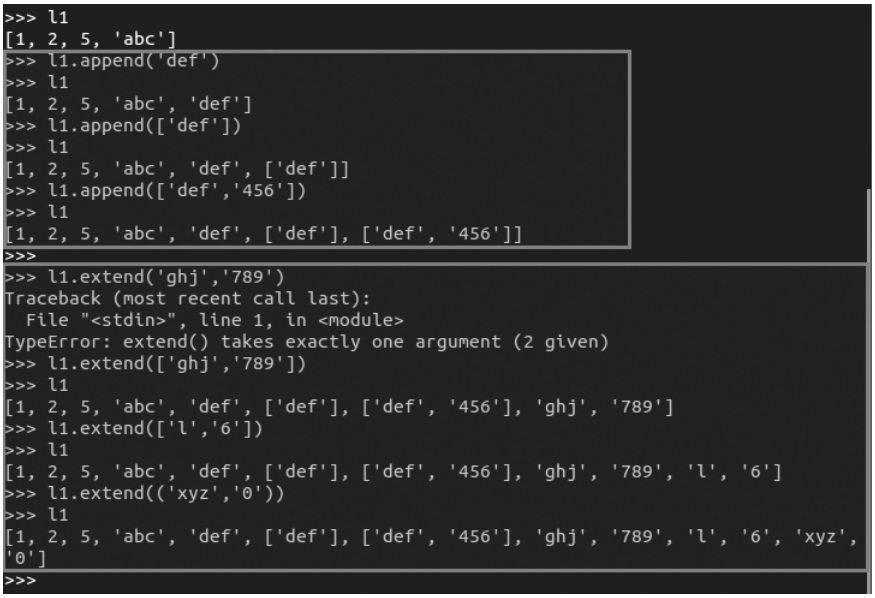


图 1-51　用 append 和 extend 方法修改列表元素

在该实例中可以看出，append 只是将传递给它的参数原封不动地附加到调用它的列表的末尾，而 extent 的参数是一个序列，将序列中的元素取出，依次加到列表的末尾，注意只取一层，如果取出的元素是列表，则直接添加到末尾，不再进一步取出。

（5）列表支持 insert、remove、pop 等方法进行操作。这些方法通过实例掌握起来并不

困难。下面先给出各种方法的说明，然后给出一个使用这些方法的实例。

列表支持利用 insert 方法向指定的索引位置插入一个元素。插入后，该元素就会占据该索引的位置。该索引位置原先的元素和其后面的元素都会相应地向后移动一个位置，相应地索引值也都会加 1。使用 remove 方法可以在列表中删除一个元素，待删除的元素作为参数传递给 remove 方法。注意，如果传递给 remove 的元素在列表中有重复，则只删除第一个。pop 方法可以从列表中弹出一个元素，在不增加参数时，即默认情况，是从末尾弹出元素，当给出索引参数时，会将索引位置的元素弹出。pop 方法的返回值是弹出的元素，同时列表也会相应地改变。此外，Python 的 del 函数可以删除列表中指定索引位置的元素，如图 1-52 所示。

```
>>> l2=['a','b','c']
>>> l2.insert(1,'x')
>>> l2
['a', 'x', 'b', 'c']
>>> l2.insert(1,'b')
>>> l2
['a', 'b', 'x', 'b', 'c']
>>> l2.remove('b')
>>> l2
['a', 'x', 'b', 'c']
>>> p1=l2.pop()
>>> p1
'c'
>>> l2
['a', 'x', 'b']
>>> p2=l2.pop(1)
>>> p2
'x'
>>> l2
['a', 'b']
>>> del l2[1]
>>> l2
['a']
>>>
```

图 1-52　列表的 insert、remove、pop 等操作

对于一些边界情况，这些方法也有相应的处理，如 insert 方法，当给出的索引超出列表已有的索引范围时并不会产生异常。如果索引大于最右端的索引值，则将元素插入到最右端，即最后一个元素的位置；如果索引小于最左端的索引值，则将元素插入到列表的最左端，即第一个元素的位置，如图 1-53 所示。

```
>>> l2.insert(1,'b')
>>> l2
['a', 'b']
>>> l2.insert(1,'b')
>>> l2
['a', 'b', 'b']
>>> l2.insert(10,'b')
>>> l2
['a', 'b', 'b', 'b']
>>> l2.insert(-100,'b')
>>> l2
['b', 'a', 'b', 'b', 'b']
>>>
```

图 1-53　insert 边界情况的处理

列表是重要的数据结构，感兴趣的读者可以进一步查阅相关的技术手册和书籍来深入了解这些内容。

（6）对列表排序可以使用列表方法 sort( )，也可以使用 Python 内置的 sorted 函数。图 1-54 给出使用 sort 方法的简单实例。

```
>>> l3=[2,1,5,7]
>>> l3
[2, 1, 5, 7]
>>> l3.sort()
>>> l3
[1, 2, 5, 7]
>>> l3.sort(reverse=True)
>>> l3
[7, 5, 2, 1]
>>>
```

图 1-54　sort 方法的使用

在该实例中，sort 方式实现了对 l3 列表的升序排序，当向 sort 传入 reverse = True 参数时，表示逆序排序。

除了 reverse 参数外，sort 还有 key 参数，图 1-55 给出 key 的使用方法。

```
>>> x = ['hello','abc','1234']
>>> x.sort()
>>> x
['1234', 'abc', 'hello']
>>> x.sort(key=len)
>>> x
['abc', '1234', 'hello']
>>> x = ['hello','abc','1234']
>>> x.sort(key=lambda elem:len(elem))
>>> x
['abc', '1234', 'hello']
>>>
```

图 1-55　key 的使用方法

在该实例中，x. sort(key = len) 表示在对 x 进行排序，并在列表元素之间进行比较时，比较的依据是使用函数 len 作用于各元素的结果，也就是说，是按照元素的长度进行比较排序的，可以根据需要使用不同的函数。注意，key = 函数名。该函数接受一个参数，不能写成函数的调用形式，如写成 key = len( ) 是不对的。

另一种指定 key 的方式是使用匿名函数。匿名函数又称 lambda 算子。在上面的实例中

```
key = lambda elem:len(elem)
```

其中，lambda elem:len(elem) 就是 lambda 算子。它表示一个函数。这个函数接受一个参数，参数名为 elem。参数名可以是符合命名规则的任意名称。冒号后面是函数体，表示函数执行的动作。其含义是在进行列表元素的比较时，比较的依据是将各元素分别传入 key 代表的 lambda 算子，然后根据算子的执行结果作为比较的依据。这与 key = len 的效果是一样的。

（7）列表推导

列表推导（list comprehensions）是一种常用的产生列表的方式，如图 1-56 所示。

```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> l = [x for x in range(1,101) if x%2==0]
>>> l
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82,
 84, 86, 88, 90, 92, 94, 96, 98, 100]
>>>
```

图 1-56　列表推导的使用方法

生成的列表中包含了 1 ~ 100 之间的偶数。

1.10　字典

字典是一种常用的数据结构，又称其为关联数组。字典和列表与元组索引方式是不同的。字典使用键（key）进行索引。列表和元组使用整数进行索引。与键对应的是值（value）。字典是由若干“键值对”组成的，可以根据键访问键所对应的值，语法为“字典名[键]”，如在向服务器发起请求时，请求头文件可以使用字典结构表示。比如，对 http://www.phei.com.cn/的请求头可以使用一个字典描述。其中，Host 键对应的值为 www.phei.com.cn，如图 1-57 所示。

```
>>> header = {
...     'Host': 'www.phei.com.cn',
...     'User-Agent': 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0) Gecko/20
100101 Firefox/44.0',
...     'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0
.8',
...     'Accept-Language': 'en-US,en;q=0.5',
...     'Accept-Encoding': 'gzip, deflate',
...     'Connection': 'keep-alive'
... }
>>> header['Host']
'www.phei.com.cn'
>>>
```

图 1-57　字典的表示方法

“Host”是字典 header 的一个键，对应的值可以用 header["Host"]访问。

也可以使用{}定义一个空的字典，里面不包含任何键值对，以后可以向字典中增加键值对，如图 1-58 所示。

如果希望根据键名获得键值，在操作中使用了字典中不存在的键名，就会触发 KeyError 异常，如图 1-59 所示。

图 1-58　向字典中增加键值对

图 1-59　字典访问异常

可以使用字典的 get( )方法来解决这类问题。get( )接受两个参数：第一个是要查的键名；第二个是字典中无此键名时的返回值，默认值为 None，如图 1-60 所示。

图 1-60　get 方法的使用

注意，字典的 get 方法可以返回一个默认值，但不会对字典产生影响，如果查找的键名不存在，不会将此键名和默认的键名放入字典中，最后 empty_d 的内容并没有改变。

字符串、整数类不可变类型都可以作为字典的键，元组也可以作为键，前提是元组的元素都是不可变类型。如果元组中直接或间接包含可变类型，则不可以作为键来使用。

与列表推导（list comprehensions）一样，字典也有相应的字典推导（dict comprehensions），如图 1-61 所示，生成了 10 个链接，以 1 ~ 10 的 10 个数字作为键，以键为页号的 url 作为值。

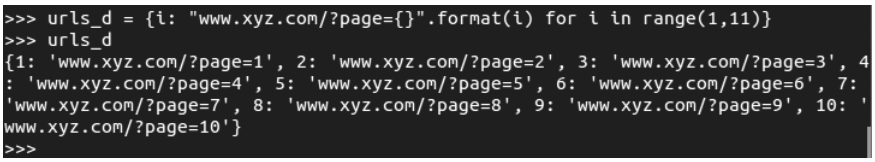


图 1-61　字典推导

可以使用字典的 items( )方法对字典进行循环遍历，每次可以同时获得键和值；或者可以使用 keys( )方法对键进行遍历，每次可以利用键获得相应的值。两种方法类似。下面是两种方法的对比。

```
>>> for key,val in urls_d.items():
...     print(key,val)
...
1 www.xyz.com/?page=1
2 www.xyz.com/?page=2
3 www.xyz.com/?page=3
4 www.xyz.com/?page=4
5 www.xyz.com/?page=5
6 www.xyz.com/?page=6
7 www.xyz.com/?page=7
8 www.xyz.com/?page=8
9 www.xyz.com/?page=9
10 www.xyz.com/?page=10
```

```
>>> for key in urls_d.keys():
...     print(key,urls_d[key])
...
1 www.xyz.com/?page=1
2 www.xyz.com/?page=2
3 www.xyz.com/?page=3
4 www.xyz.com/?page=4
5 www.xyz.com/?page=5
6 www.xyz.com/?page=6
7 www.xyz.com/?page=7
8 www.xyz.com/?page=8
9 www.xyz.com/?page=9
10 www.xyz.com/?page=10
```

1.11 集合

集合的特点是无序且无重复元素。Python 提供了集合类型 set 和相关的操作，主要包括成员测试、添加元素、去除元素等方法，也提供了数学中常见集合的交、并、差运算等。

可以使用 set()或大括号{}来初始化集合，如图 1-62 所示。

```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> s1=set(['a','b','c'])
>>> s1
{'b', 'a', 'c'}
>>> s2={'a','b','c'}
>>> s2
{'b', 'a', 'c'}
>>> s1==s2
True
>>>
```

图 1-62　集合的初始化

空集合只能使用 set()来定义，当{}中没有写任何内容的时候，其实创建的是一个空的字典。图 1-63 对两者进行了比较。

可以使用 add()方法向集合中添加一个元素，或者使用 update()方法向集合中添加多个元素。判断一个元素是否在一个集合中可以使用 in 操作，如果元素在集合中，则返回 True，否则返回 False，如图 1-64 所示。

```
>>> s3={}
>>> type(s3)
<class 'dict'>
>>> s4=set()
>>> s4
set()
>>> type(s4)
<class 'set'>
>>>
```

图 1-63　空集合的定义方式

```
>>> s4.add('x')
>>> s4
{'x'}
>>> s4.update(['y','z'])
>>> s4
{'y', 'z', 'x'}
>>> 'y' in s4
True
>>>
```

图 1-64　add、update 和 in

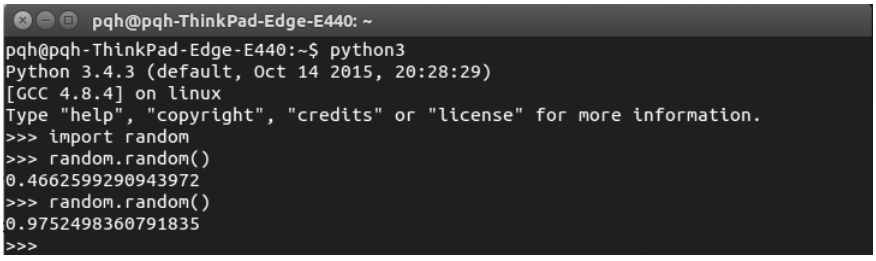
可以使用循环遍历集合中的元素，如图 1-65 所示。

```
>>> for x in s4:
...   print(x)
...
y
z
x
>>>
```

图 1-65　集合中元素的遍历

1.12　随机数

可以利用 Python 的 random 模块中的方法生成随机数，如 random. random 将生成 0 ~ 1 之间的随机数，如图 1-66 所示。



```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> random.random()
0.4662599290943972
>>> random.random()
0.9752498360791835
>>>
```

图 1-66　random 模块的使用

可以使用 random. randint(a,b)方法返回 a ~ b 之间（包括 a，b）的整数，如图 1-67 所示。

```
>>> random.randint(2,6)
4
>>> random.randint(2,6)
5
>>> random.randint(2,6)
4
>>> random.randint(2,6)
6
>>> random.randint(2,6)
4
>>>
```

图 1-67　random. randint 方法的使用

如果希望可以在程序中多个产生随机数的位置产生相同的随机数，则可以通过设置随机数的种子值来实现，如图 1-68 所示。

```
>>> random.random()
0.3176824848078029
>>> random.seed(20)
>>> random.random()
0.9056396761745207
>>> random.random()
0.6862541570267026
>>> random.random()
0.7665092563626442
>>> random.seed(20)
>>> random.random()
0.9056396761745207
>>>
```

图 1-68　random. seed 方法的使用

1.13　enumerate 的使用

可以使用 enumerate 在对列表、元组等类型操作时，同时返回元素的索引和元素的值，如图 1-69 所示。

```
pqh@pqh-ThinkPad-Edge-E440: ~
pqh@pqh-ThinkPad-Edge-E440:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> for ind, val in enumerate([10, 20, 30, 40,50]):
...    print(ind,val)
...
0 10
1 20
2 30
3 40
4 50
>>>
```

图 1-69　enumerate 的基本用法

输出索引时，enumerate 默认是从 0 开始索引，可以在 enumerate 中增加一个参数，表示第一个索引的值，如图 1-70 所示。

```
>>> for ind, val in enumerate([10,20,30,40,50],1):
...   print(ind, val)
...
1 10
2 20
3 30
4 40
5 50
>>>
>>> for ind, val in enumerate([10,20,30,40,50],3):
...   print(ind, val)
...
3 10
4 20
5 30
6 40
7 50
>>>
```

图 1-70　enumerate 首元素索引的设置方法

1.14　第二个实例

下面给出本章的第二个重要实例。在该实例中，将字典、列表、元组、集合、循环、异常、文件操作等融合在一起，代码如下，保存在 third_get. py 文件中。

```
import requests

urls_dict = {
    '电子工业出版社': http://www.phei.com.cn/',
    '在线资源': http://www.phei.com.cn/module/zygl/zxzyindex.jsp',
    'xyz': www.phei.com.cn',
    '网上书店 1':
        'http://www.phei.com.cn/module/goods/wssd_index.jsp',
    '网上书店 2':
        'http://www.phei.com.cn/module/goods/wssd_index.jsp'
}

urls_lst = [
    ('电子工业出版社', http://www.phei.com.cn/'),
    ('在线资源', http://www.phei.com.cn/module/zygl/zxzyindex.jsp'),
    ('xyz', www.phei.com.cn'),
```

```
    ('网上书店1',
     'http://www.phei.com.cn/module/goods/wssd_index.jsp'),
    ('网上书店2',
     'http://www.phei.com.cn/module/goods/wssd_index.jsp'),

]
#利用字典抓取
crawled_urls_for_dict = set()
for ind,name in enumerate(urls_dict.keys()):
    name_url = urls_dict[name]
    if name_url in crawled_urls_for_dict:
        print(ind,name,"已经抓取过了.")
    else:
        try:
            resp = requests.get(name_url)
        except Exception as e:
            print(ind,name,':',str(e)[0:50])
            continue
        content = resp.text
        crawled_urls_for_dict.add(name_url)
        with open('bydict_'+name+'.html','w',encoding='utf8') as f:
            f.write(content)
            print("抓取完成:{} {},内容长度为{}".format(
                                                ind,name,len(content)))

for u in crawled_urls_for_dict:
    print(u)

print('-'*60)

#利用列表抓取
crawled_urls_for_list = set()
for ind,tup in enumerate(urls_lst):
    name = tup[0]
    name_url = tup[1]
    if name_url in crawled_urls_for_list:
```

```
            print(ind,name," 已经抓取过了.")
        else:
            try:
                resp = requests.get(name_url)
            except Exception as e:
                print(ind,name,': ',str(e)[0:50])
                continue
            content = resp.text
            crawled_urls_for_list.add(name_url)
            with open('bylist'+name+'.html', 'w',encoding='utf8') as f:
                f.write(content)
                print("抓取完成:{} {},内容长度为{}".format(
                                                    ind,name,len(content)))

for u in crawled_urls_for_list:
    print(u)
```

该段代码演示了两种抓取方式。一种是将待抓取的目标信息组织成字典，名为 urls_dict。其中，使用键来表示目标的名字，相应的值表示抓取目标的网址。字典中共包含 5 个目标信息。其中，'网上书店 1'和'网上书店 2'的网址信息是相同的；键为'xyz'的项是特意加上去的，注意网址和其他网址的区别，即没有“http://”作为前缀，对于 requests 库来说是无效的网址，抓取时将抛出异常。这也是引入该项的目的，可演示抓取时异常处理的情况。

另一种抓取方式是将待抓取的目标信息组织成列表，名为 urls_lst。列表的元素是元组，使用元组来表示目标信息。每个元组包含两个元素：第一个元素是抓取目标的名称；第二个元素是抓取目标的网址。列表中共有 5 个元组，表示的内容信息与 urls_dict 中的完全一样。

使用两种方式组织信息的主要目的是演示使用 for 循环结构对不同数据类型的遍历方法，在实际使用中使用列表的情况会更多些。

在代码中定义了两个集合，即 crawled_urls_for_dict 和 crawled_urls_for_list，分别用于记录在两种方式下抓取的网址。集合中是不能有重复元素的，可以达到去重的效果。在后面的代码中可以看到，本例中的去重方法利用列表也可以完成，使用集合主要是演示其基本的使用方式。

下面首先介绍第一种利用字典的方式。在进行循环前，首先执行 urls_dict.keys()从字典中获得键的列表，由于字典的无序性，因此使用 keys 方法获得键的列表时，可能键在列

表中的排列顺序会发生变化，获得键的列表后再使用 enumerate 函数，就可以在 for 循环遍历的同时获得键 name 和在列表中相应的索引 ind，接下来

```
name_url = urls_dict[ name ]
```

根据 name 在字典中获得该 name 对应的网址 name_url 作为待抓取的地址。在抓取之前，首先判断该地址是否被抓取过，如果已经被抓取，则跳过，进行下一个循环抓取下一个网址；如果没有被抓取过，则进行抓取，并将抓取到的内容保存到文件中。因为抓取过的地址保存在 crawled_urls_for_dict 集合中，因此在判断时使用的是

```
name_url in crawled_urls_for_dict
```

name_url 在集合 crawled_urls_for_dict 中返回 Ture，否则返回 False。在 Python 中使用 in 可以判断左面元素是否存在于右边元素中，in 的使用比较灵活，也可以用于判断某个字符是否存在于一个字符串中，或者某个元素是否存在于列表中等多种情况。

如果结果为 True，则表示已经抓取过。当结果为 False 时，利用 requests 的 get 方法进行抓取。抓取时可能会遇到各种异常情况，导致程序因异常而终止，因此将

```
resp = requests. get( name_url )
```

置于异常处理结构中。从代码中可以看到，当发生异常时，会输出异常发生时正在抓取的 name 在 name 列表中的索引、抓取的 name 及异常内容。有时异常信息会特别长，这里只输出异常信息的前 50 个字符。因为 e 代表异常对象，所以先使用 str(e) 获得 e 的字符串信息，然后进行切片输出。产生异常的原因有多种，为了触发异常，本例中在 urls_dict 中特别放入了一项' xyż ': www. phei. com. cń 。其中，网址的形式是 requests 不能处理的，因此会触发一个异常，在后面运行时会看到异常信息。

如果没有异常发生，则从抓取返回的响应对象 resp 中得到 text 属性，即响应的文本信息内容，将其赋给 content，是 content = resp. text 需要完成的工作。下一句

```
crawled_urls_for_dict. add( name_url )
```

的作用是将抓取的网址放入 crawled_urls_for_dict 集合中，表示这是一个抓取过的网址。根据本例中设计的抓取机制，在下次循环进行 name_url in crawled_urls_for_dict 判断时，如果 name_url 已在集合中，就不会再抓取了。

for 循环最后的部分就是将本次循环的抓取结果进行保存，在方式 1 下根据字典结构抓取，保存的文件名加了 bydict 前缀，以区分后面在方式 2 下基于列表的抓取方式。保存完成后，输出本次抓取的相应信息。

方式 1 最后使用 for 循环结构对 crawled_urls_for_dict 进行遍历，输出已抓取过的网址。

最后输出由 60 个“ - ”组成的字符分割串。下面是方式 2 利用列表进行抓取的实例，使用 crawled_urls_for_list 集合收集已经抓取过的网址信息。剩下的 for 循环部分与方式 1 代码和功能基本一致，主要的差异在 for 循环的开始部分，使用

```
enumerate(urls_lst)
```

来获得 urls_lst 待循环遍历的元素索引 ind 和元素 tup。根据前面的介绍可知，tup 的第一个元素 tup[0]就是方式 1 中的 name，tup[1]就是方式 1 中的 name_url。这里仍保持相同的命名。接下来的逻辑就与方式 1 中一致了。

third_get. py 的运行结果如图 1-71 所示。

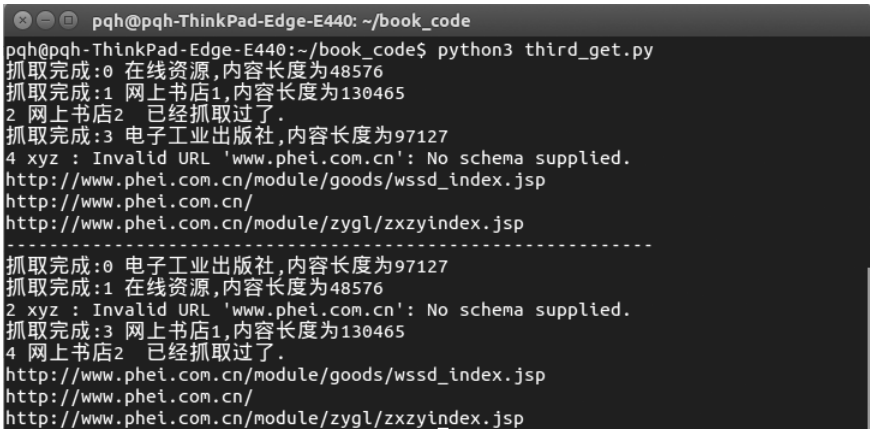


图 1-71　third_get. py 的运行结果

注意，在方式 1 中，for 循环遍历的列表是由 urls_dict. keys( )获得的，由于字典的无序性，因此运行时获得 name 列表的元素顺序可能有所不同，运行时出现的“抓取完成”信息的顺序也可能不同。

实战

安装 Ubuntu 14. 04 操作系统或使用虚拟机安装该系统，运行本章中的两个程序。

2 第 2 章 字符串解析

本章介绍 Python 处理字符串的基本方法，包括 Python 字符串类型自带的字符串方法、正则表达式、利用 BeautifulSoup 抽取 HTML 文件及利用 Python 的 json 模块处理 json 格式的字符串。这些方法和模块在数据抓取过程中为抓取内容字符串的形式转化和信息抽取提供了灵活和强大的支持，简化了对问题的处理。在介绍 json 格式的数据处理时，介绍了使用 MongoDB 数据库存储 json 格式数据和展示数据内容的方法。

2.1 常用函数

进行字符串分析时，可以使用 Python 提供的基本字符串方法对字符串进行简单处理，也可以使用正则表达式模块应对更为复杂的情况。本节介绍几个基本的字符串处理方法。下一节介绍正则表达式的相关知识。

基本的字符串处理方法包括 split( )、replace( )、strip( )等。split 方法可以将一个字符串按照指定的标识分割成几部分。这几部分将以列表（list）的形式返回，由 split 返回的列表是由字符串元素组成的，在交互环境下输入 help(str. split)的结果如图 2-1 所示。

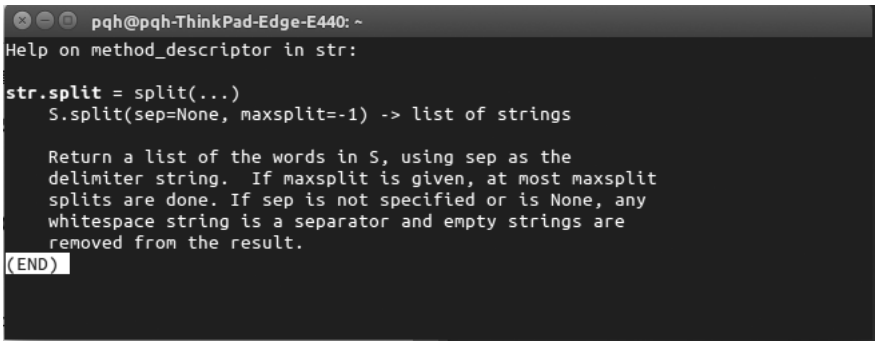


图 2-1 str. split 的使用帮助

该函数最多接受两个参数。其中，sep 代表分割字符串时基于的分割符，maxsplit 表示分割的次数。下面先给出 split 函数较为常用的使用方式，然后结合这两个参数的默认值给出较为特殊的使用情况，如图 2-2 所示。

```
>>> "a/b/c".split("/")
['a', 'b', 'c']
>>> "aa@@bb@@cc".split("@@")
['aa', 'bb', 'cc']
>>>
```

图 2-2　split 的使用 1

如果希望取出某一部分，则在返回的列表中利用相应部分的索引即可，如在“a/b/c”中希望得到按“/”分割第二部分 b，如图 2-3 所示。

```
>>> "a/b/c".split("/")[1]
'b'
>>>
```

图 2-3　split 的使用 2

split 的参数应为长度大于 0 的有效字符串，不可以为""。如果参数字符串没有在待分割字符串中出现，则无法分割，这时仍将返回一个列表，列表中只有一个元素，就是待分隔的字符串。如果参数字符串位于待分割字符串的左端或右端，则在分割的结果中出现""，如图 2-4 所示。

```
>>> "a/b/c".split("")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: empty separator
>>> "a/b/c".split("c")
['a/b/', '']
>>> "a/b/c".split("+")
['a/b/c']
>>> "aa".split("aa")
['', '']
>>>
```

图 2-4　split 的使用 3

对于出现""的情况，可以想像成一个字符串 s == "" + s + ""，如图 2-5 所示。

```
>>> es = ""
>>> s = es+"1"
>>> s
'1'
>>> s==es+"1"
True
>>> s=="1"+es
True
>>> s==es+"1"+es
True
>>>
```

图 2-5　""的处理

对于 split 的 maxsplit 参数表示使用 sep 对目标字符串分割的次数，默认值为 -1。如果目标字符串中不含 sep 分隔字符串，则分割结果为只含有目标字符串的列表，与 maxsplit 的取值无关。假设目标字符串中含有 seq 分割字符串，则 maxsplit 的值取默认值为 -1，可对目标串进行完全分割。当 maxsplit 为 0 时，表示不分割；若 maxsplit 为大于 1 的整数时，会按照 maxsplit 指定的次数进行分割；当 maxsplit 的值大于 seq 在目标字符串中出现的次数时，则作为完全分割处理，如图 2-6 所示。

```
>>> expr = '1+2+3+4+5'
>>> expr.split('xxx', 7)
['1+2+3+4+5']
>>> expr.split('xxx', -1)
['1+2+3+4+5']
>>> expr.split('+', 0)
['1+2+3+4+5']
>>> expr.split('+', -1)
['1', '2', '3', '4', '5']
>>> expr.split('+', 1)
['1', '2+3+4+5']
>>> expr.split('+', 2)
['1', '2', '3+4+5']
>>> expr.split('+', 7)
['1', '2', '3', '4', '5']
>>>
```

图 2-6　split 的使用 4

当 seq 为 None 默认值，也就是说，在分割时不指定分割字符串时，会按目标串中的空白符进行分割，空白符号包括空格、制表符（“\t”）或换行（“\n”）等。分隔时所有的空白都会被分割，且多个连续的空白会被作为一个空白进行分割，如图 2-7 所示。

```
>>> "1 3\t4   5\n\t67  ".split()
['1', '3', '4', '5', '67']
>>> "1 3\t4   5\n\t67  ".split(maxsplit=2)
['1', '3', '4   5\n\t67  ']
>>>
```

图 2-7　split 的使用 5

注意在这种情况下，不会产生前面提到的在左右边界分隔时产生的""，如图 2-8 所示。

```
>>> "1 2 3 ".split()
['1', '2', '3']
>>> "1 2 3 ".split(' ')
['1', '2', '3', '']
>>>
```

图 2-8　split 的使用 6

replace 主要用于字符串的取代操作，在交互界面可以查看 replace 的帮助文档，在交互环境下输入 help（str. replace），帮助内容如图 2-9 所示。

该函数可以对目标字符串中的特定字符串（old）使用新字符串（new）进行替换，并将替换后形成的字符串返回。[，count] 表示 count 是一个可选参数，如果函数中不使用 count 作为参数，则会对目标字符串中出现的全部 old 进行替换，否则将只替换字符串前面出现的 count 个 old，如图 2-10 所示。

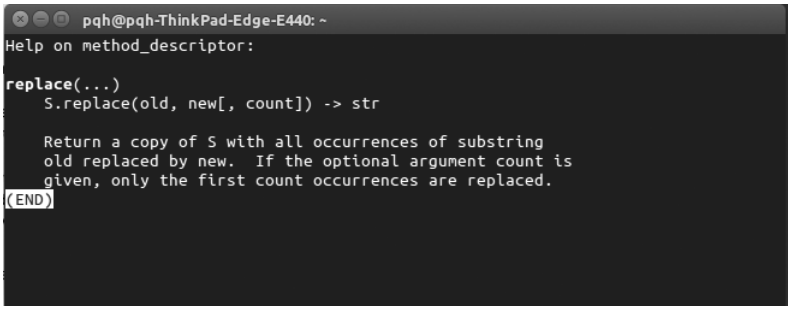


图 2-9　replace 的帮助文档

```
>>> 'a123a123a123'.replace('a','!')
'!123!123!123'
>>> 'a123a123a123'.replace('a','!',0)
'a123a123a123'
>>> 'a123a123a123'.replace('a','!',1)
'!123a123a123'
>>> 'a123a123a123'.replace('a','!',2)
'!123!123a123'
>>> 'a123a123a123'.replace('a','!',10)
'!123!123!123'
>>> 'a123a123a123'.replace('a','!',-1)
'!123!123!123'
>>>
```

图 2-10　replace 的使用 1

注意后面两种情况，当 count 的值大于 old 在目标串中出现的次数时，会对全部 old 进行替换；如果 count 是负数，也进行全部替换。

如果在目标串中不包含 old 指定的字符串，则不进行替换，如图 2-11 所示。

```
>>> 'a123a123a123'.replace('b','!')
'a123a123a123'
>>>
```

图 2-11　replace 的使用 2

因为 replace 函数的返回值是 str 类型，因此可以连续使用 replace 实施字符串的替换，如图 2-12 所示。

```
>>> 'a123a123a123'.replace('b','!').replace('a','@')
'@123@123@123'
>>>
```

图 2-12　replace 的使用 3

strip 也是常用的字符串处理函数，在交互环境下输入 help(str. strip)，可以查看 strip 的帮助，如图 2-13 所示。

strip 可以将目标字符串中行首和行尾的空白（包括空格(' ')、制表符(\t)、换行(\n)）去掉，[chars]表明 chars 是可选参数，当指定 chars 时，会将行首和行尾的 chars 同时去掉，如图 2-14 所示。

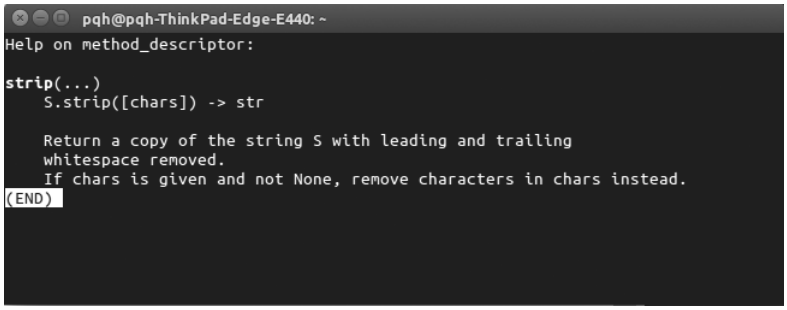


图 2-13　strip 的帮助文档

图 2-14　strip 的使用 1

可见，目标字符串的首尾空白都被去掉了，而且是连续地去掉，直至遇到第一个非空白字符为止。图 2-15 给出带参数的形式。

图 2-15　strip 的使用 2

在前两种情况中，chars 参数只出现在一边，因此只在一边进行了去除。后一种情况“l”出现在两边，因此都会被去除，而且与去除空白的模式类似，在字符串两边都会去除至不再出现“l”为止。因为 strip 函数的返回类型是 str，因此可以连续使用 strip 进行去除，并与 replace 等结合使用，如图 2-16 所示。

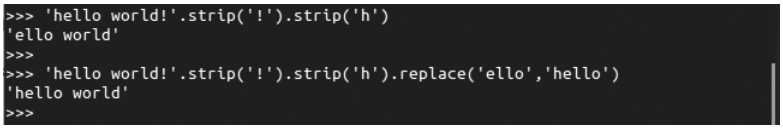


图 2-16　strip 的使用 3

2.2　正则表达式

正则表达式是处理字符串的有力工具。Python 的 re 模块提供了大量的方法，实现了正则表达式相关的各类操作。本节将通过具体实例介绍 Python 中正则表达式的基本使用方法，当遇到其他问题时，可以在此基础上查找手册，通过学习实例的方法加以解决。

在交互命令行下可以查看 re 模块的帮助，在交互环境下输入 help('re')，在交互界面下显示 re 模块的帮助文档。其中，FUNCTIONS 的部分给出了 re 模块中主要的方法，前三个方法如图 2-17 所示。

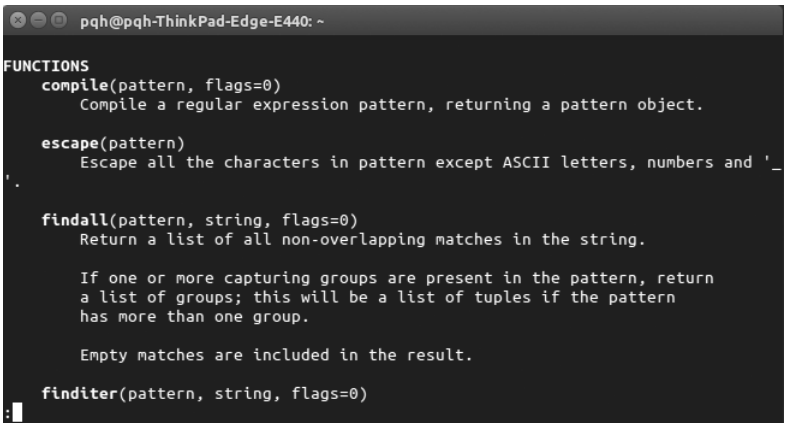


图 2-17　re 帮助文档的 FUNCTIONS 部分

在使用和学习的过程中会遇到关于正则表达式的各种问题，可以利用各种方法和它们之间的组合解决。在上面给出的方法中，第一个参数都是 pattern。那么什么是 pattern 呢？

pattern 又称模式，在上面帮助中列出的方法中，参数中的 pattern 是一个字符串。pattern 的特殊之处在于其中可以包含一些特殊的字符。这些字符或组合可以作为模式去匹配更多与其表示特征相同的字符串。最常见的模式就是通配符，如“+”“*”“?”“.”等。

常用的正则表达式字符和通常的含义见表 2-1。更为详细的使用说明见 Python3.4 关于正则表达式的在线文档，网址为 https://docs.python.org/3.4/library/re.html。

表 2-1　常用的正则表达式字符和通常的含义

符　号	含　义	符　号	含　义
.	匹配除“\n”外的任意字符	\s	匹配空白字符，等同于[\t\n\r\f\v]。注意，\t 前是空格符
+	匹配前一个字符 1 次或多次	\S	匹配任何非空白字符，等同于[^\t\n\r\f\v]
*	匹配前一个字符 0 次或多次	\d	匹配数字，等同于[0-9]
?	匹配前一个字符 0 次或 1 次	\D	匹配任何非数字，等同于[^0-9]
()	括号括起来的表达式表示一个分组	\w	匹配单词字符(word characters)，等同于[a-zA-Z0-9_]
^	匹配字符串的开头	\W	匹配任何非单词字符 等同于[^a-zA-Z0-9_]
$	匹配字符串的末尾		
[]	表示字符的集合，可用-表示范围；当^出现在[]第一个字符时表示取反		

下面通过实例给出正则表达式的一些简单使用方法，包括 match、search、findall 等。

首先介绍 match( )方法。该方法接受两个参数：第一个参数是模式字符串；第二个参数是一个字符串。该方法使用模式字符串从第二个参数给定的字符串的起始位置进行匹配，如果匹配成功，则返回一个 match 对象，否则返回 None。其实例如图 2-18 所示。

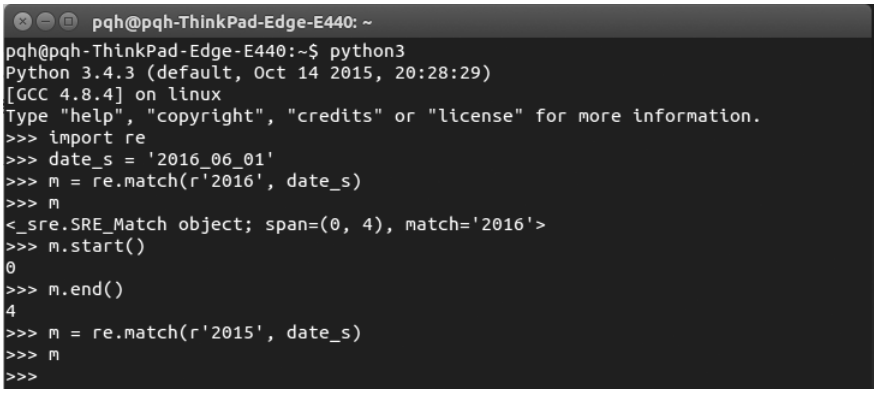


图 2-18　match 的使用实例 1

在该实例中，模式字符串分别为“2016”和“2015”，在两个字符串中并没有出现特殊的模式匹配字符，从 date_s 字符串的最左端开始匹配。显然，“2016”是可以匹配的，匹配的结果赋给了 m。从图中可以看到，m 是一个对象，m 的 start( )和 end( )方法分别给出了匹配字符串的开始索引和结束索引。字符串“2016”匹配失败，m 为 None，体现在交互环境下就是在提示符后输入 m 而没有任何输出。

在 r'2016' 和 r'2015' 前面的符号 r 是 Python 的“raw string”的标识。如果在一个字符串前面出现 r 作为前缀，则反斜杠不会被特殊处理。比如，r“\n”表示两个字符的字符串包括“\”和“n”，而不加前缀 r 的“\n”是一个单字符字符串表示一个换行。本实例的' 2015' 和' 2016' 中没有“\”出现，也加上了 r 前缀，不会有所影响。

下面给出在模式字符串中使用正则表达式符号的实例。仍以 date_s 为例，这个字符串可以认为是由下划线连接的年、月、日组成的日期字符串。年月日都是由数字组成的字符串，r'\d' 是一个正则表达式，可以匹配任何一个十进制数字字符，相当于 r'[0-9]'，'+' 表示匹配前一个字符一次或多次，那么 r'\d+' 就表示匹配长度至少为 1 的，由 0~9 数字字符表示的字符串，因此可以用

```
r'\d+_\d+_\d+'
```

这个正则表达式作为模式匹配 date_s，如图 2-19 所示。

前面表格中的( )可以包含一个正则表达式，此时被认为是一个分组，在正则表达式中使用分组，在实现匹配功能的基础之上，分组中匹配的内容可以抽取出来，方便后续的使

用。下面将 r‘ \d + _\d + _\d + ’进一步处理，分别将表示年、月、日的正则表达式使用分组的方式表示，然后进行匹配，并将匹配的结果抽取出来，如图 2-20 所示。

```
>>> m = re.match(r'\d+_\d+_\d+', date_s)
>>> m
<_sre.SRE_Match object; span=(0, 10), match='2016_06_01'>
>>> m.start()
0
>>> m.end()
10
>>> len(date_s)
10
>>>
```

图 2-19　match 的使用实例 2

```
>>> m = re.match(r'(\d+)_(\d+)_(\d+)',date_s)
>>> m
<_sre.SRE_Match object; span=(0, 10), match='2016_06_01'>
>>> m.group(0)
'2016_06_01'
>>> m.group(1)
'2016'
>>> m.group(2)
'06'
>>> m.group(3)
'01'
>>> m.groups()
('2016', '06', '01')
>>>
```

图 2-20　match 的使用实例 3

分组在正则表达式中从左到右是从 1 开始计数的，左边的第一个分组就是第 1 个分组，而不是第 0 个分组。可以通过调用匹配对象的 group 方法获得分组的内容，当 group 的参数为 0 时，返回值是全部匹配字符串，参数为 1 时，返回第一个分组的内容，参数为 2 时，返回第二个分组的内容，依此类推。这些结果可以从如图 2-20 所示中看出。方法 groups 返回一个包含全部匹配分组内容的元组。

match 方法只能从字符串的开始进行匹配，search 方法会扫描整个字符串查找匹配，两者对比如图 2-21 所示。

```
>>> date_s_2 = 'today is 2016_06_01'
>>> m_match = re.match(r'(\d+)_(\d+)_(\d+)',date_s_2)
>>> m_match
>>> m_search = re.search(r'(\d+)_(\d+)_(\d+)',date_s_2)
>>> m_search
<_sre.SRE_Match object; span=(9, 19), match='2016_06_01'>
>>> m_search.group(0)
'2016_06_01'
>>> m_search.group(1)
'2016'
>>> m_search.group(2)
'06'
>>> m_search.group(3)
'01'
>>> m_search.groups()
('2016', '06', '01')
>>>
```

图 2-21　match 和 search 的使用对比

在该实例中，字符串变为了‘today is 2016_06_01’，使用前面使用过的 match 进行日期的匹配，结果赋给 m_match，从图中交互环境的运行结果可以看出 m_match 为 None，而 search 可以正确地匹配日期。

findall 方法可以找到所有的匹配，结合下面的实例来说明这个问题。在这个实例中的字符串为

```
url = http://www.xyz123.com/prod_lst?start_time=2016_01_01&end_time=2016_12_31
```

这是一个假设的 url。假设通过这个 url 可以获得从 2016_01_01 开始至 2016_12_31 截止这段时间内的产品列表。如果仍使用 r'(\d+)_(\d+)_(\d+)' 作为模式，则使用 match 方法是无法匹配的。因为 match 是从字符串的左边开始匹配，此时可使用 findall 方法。该方法能在字符串中找到所有匹配，并以列表的形式返回。如果没有匹配到，则返回空列表，如图 2-22 所示。

```
>>> url='http://www.xyz123.com/prod_lst?start_time=2016_01_01&end_time=2016_12_3
1'
>>> m=re.match(r'(\d+)_(\d+)_(\d+)', url)
>>> m
>>>
>>> m=re.findall(r'(\d+)_(\d+)_(\d+)', url)
>>> m
[('2016', '01', '01'), ('2016', '12', '31')]
>>>
```

图 2-22　find_all 的使用方法

除了直接将模式字符串作为参数传入到相应的方法之外，还可以利用 compile 方法将模式字符串编译成模式对象，再通过模式对象提供的方法来完成匹配，如图 2-23 所示。

```
>>> pattern_obj=re.compile(r'(\d+)_(\d+)_(\d+)')
>>> pattern_obj.findall(url)
[('2016', '01', '01'), ('2016', '12', '31')]
>>>
```

图 2-23　正则对象的使用方法

在该实例中，pattern_obj 是一个模式对象，该对象也具有 findall 方法，调用时将待匹配的字符串传入即可。

在数据抓取时，返回的 HTML 文档返回后，一般都要将其中包含的文本抽取出来，有时抽取后可能遇到这样的情况，下面是抽取出的一行内容，即

```
Computerbook  20.00    1999    100 页
```

特点是 Computerbook 和 20.00 之间有 1 个空格，20.00 和 1999 之间有两个空格，1999 和 100 页之间有 3 个空格。

从内容看，这一行包含的信息是一本书的名称、价格、出版时间和页数，这正是我们需要的，通常可以将信息保存到一个列表中

```
[ Computerbook ', 20. 00 ', 1999 ', 100 页' ]
```

为了做到这一点，首先想到的是前面介绍的字符串方法 split，如图 2-24 所示。

```
>>> info='Computerbook 20.00  1999   100页'
>>> info.split(' ')
['Computerbook', '20.00', '', '1999', '', '', '100页']
>>>
```

图 2-24　使用 split（' ）分割的结果

显然，得到的列表中多了很多的空字符串，根据前面的介绍也可使用 split( )方法来得到希望的效果。本节给出使用正则表达式的 split 方法来完成这个工作的方式，如图 2-25 所示。

```
>>> info='Computerbook 20.00  1999   100页'
>>> re.split(r'\s+', info)
['Computerbook', '20.00', '1999', '100页']
>>>
```

图 2-25　使用 re. split 分割的结果

这个结果正是我们所需要的，与字符串的 split 方法不同，re 的 split 方法可以指定分隔符的模式。在该实例中，ŕ \s + 可以匹配一个或多个空白字符，凡是符合这个特征的字符串都会被认为是分割符，因此多个空格都可以作为一个分割符整体看待，所以得到了希望的结果。

正则表达式还支持很多方法，在使用时方法基本类似，读者可以在遇到问题时查阅相关的文档资料，学习和模仿具体的实例，进而解决问题。

2. 3　BeautifulSoup

在数据抓取的过程中，需要对抓取到的信息进行处理，很多信息是以结构化文件返回的，最常见的就是 HTML 页面文件，这是当前绝大多数网站内容的主要载体。另外，还会遇到 XML 和 json 等情况。本节主要介绍 Python 的 HTML 和 XML 的分析库 BeautifulSoup，并以 HTML 的分析为例介绍 BeautifulSoup 的使用方法。

BeautifulSoup 可以在 HTML 或 XML 结构化文档中抽取出数据，并提供各类方法，可以方便地对文档进行搜索、抽取和修改，极大地提高了工作效率。在 Ubuntu 下可以使用命令安装 BeautifulSoup，如图 2-26 所示。

```
pqh@pqh-ThinkPad-Edge-E440:~$ sudo pip2 install bs4
[sudo] password for pqh:
Downloading/unpacking bs4
  Downloading bs4-0.0.1.tar.gz
  Running setup.py (path:/tmp/pip_build_root/bs4/setup.py) egg_info for package
bs4

Downloading/unpacking beautifulsoup4 (from bs4)
  Downloading beautifulsoup4-4.4.1-py2-none-any.whl (81kB): 81kB downloaded
Installing collected packages: bs4, beautifulsoup4
  Running setup.py install for bs4

Successfully installed bs4 beautifulsoup4
Cleaning up...
pqh@pqh-ThinkPad-Edge-E440:~$ sudo pip3 install bs4
Downloading/unpacking bs4
  Downloading bs4-0.0.1.tar.gz
  Running setup.py (path:/tmp/pip_build_root/bs4/setup.py) egg_info for package
bs4

Downloading/unpacking beautifulsoup4 (from bs4)
  Downloading beautifulsoup4-4.4.1-py3-none-any.whl (81kB): 81kB downloaded
Installing collected packages: bs4, beautifulsoup4
  Running setup.py install for bs4

Successfully installed bs4 beautifulsoup4
Cleaning up...
pqh@pqh-ThinkPad-Edge-E440:~$
```

图 2-26　BeautifulSoup 的安装方法

BeautifulSoup 可以对 HTML 等结构化文档进行处理和分析，提供的多种方法可以对经其分析的文档进行各种数据的抽取操作。下面通过实例进行说明，目标网址为

http://www.phei.com.cn/module/goods/searchkey.jsp?goodtypeid=1&goodtypename=%E8%AE%A1%E7%AE%97%E6%9C%BA

网页截图如图 2-27 所示。

图 2-27　网页截图

为了方便分析，可以将页面保存至本地目录下。在页面中单击右键，在弹出的菜单中选择“查看页面源代码”，如图 2-28 所示。单击“查看页面源代码”就可以查看到该页面所对应的 HTML 代码。代码显示在一个弹出的窗口中，如图 2-29 所示。

图 2-28　页面单击右键后的菜单内容

图 2-29　页面源代码窗口

将该窗口中的代码全选、复制，然后在本地相应的目录下建立一个 beautifulSoup_test. html 的空文件，使用文本编辑器打开，然后将代码粘贴到里面并保存，就可以对保存的代码使用 BeautifulSoup 进行分析了。

下面的代码 all_a. py 实现了利用 BeautifulSoup 抽取该页面内的所有 a 标签的 href 链接和对应的文本。

```
from bs4 import BeautifulSoup
with open('beautifulSoup_set.html', 'r', encoding = 'utf8') as f:
    bs = BeautifulSoup(f.read())
    a_lst = bs.find_all('a')
    for a in a_lst:
        if a.text! = '':
            print(a.text.strip(),a['href'])
```

在 shell 中输入 python3 all_a.py 运行，运行结果如下所示。

```
......
外语 javascript:gotopage('外语');
文学 javascript:gotopage('文学');
综合 javascript:gotopage('综合');
考试与认证_大众 javascript:gotopage('考试与认证_大众');
科技 javascript:gotopage('科技');
Prezi 演示进阶之路 /module/goods/wssd_content.jsp?bookid=46449
网络安全实验教程 /module/goods/wssd_content.jsp?bookid=46439
.......
```

在代码中，使用 with 结构打开文件，这个文件就是保存首页的源代码文件，打开的文件用 f 表示，f.read()将读取文件的内容，结果是文件内文本字符串。BeautifulSoup 是一个类，我们将 f.read()返回的字符串初始化，返回的对象记为 bs。BeautifulSoup 支持很多种对文档进行操作的方法，初始化为对象后，就可以调用相应的方法进行操作了。常用的方法有 find 和 find_all。前者可以在文档中找到一个符合条件的特定元素，后者可以在文档中找到全部符合条件的元素。

由于需要在文档中找到所有 a 元素，因此使用了 find_all 方法。该方法可接受很多参数。常用的形式为

```
元素列表 = bs.find_all(元素名称,attrs = {属性名:属性值})
```

在上面的代码中，a_lst = bs.find_all('a')可以找到页面中所有的 a 元素，并以列表的形式返回。

接下来的循环结构遍历 a 元素列表中的每个 a 元素，对于每个元素返回的文本和链接值，文本就是在页面上看到的代表超链接的文字，而链接值就是单击文本时将要跳转的位置，一般存储在 a 元素的 href 属性中。在这个实例中，有些 a 元素文本是空的，我们不关心这样的元素，因此在输出时先进行了 if a.text! = '': 的判断，如果 a.text 不为空，就输出

a. text 和对应的 a 元素 href 属性的值，a. text. strip( ) 主要用于在输出 a. text 时去掉两边的空白，而 a[href]是用于取出链接值的语法。

在上面代码的输出片段中，有这样两条

```
......
Prezi 演示进阶之路 /module/goods/wssd_content. jsp? bookid = 46449
网络安全实验教程 /module/goods/wssd_content. jsp? bookid = 46439
......
```

这正是前面截图中的前两条图书的书名和链接，是一般在抓取时遇到列表页时的数据形式，通常需要跟踪这些信息进入详细页，所以上面的代码改造一下就可以变为收集当前页面图书列表相关信息的代码。下面的代码就可完成这个功能，即

```
from bs4 import BeautifulSoup
with open( beautifulSoup_set. html ', r ,encoding = utf8 ) as f:
    bs = BeautifulSoup( f. read( ) )
    a_lst = bs. find_all( a )
    for a in a_lst:
        if a. text! = and wssd_content. jsp? bookid in a[ href ]:
            print( a. text. strip( ) ,a[ href ] )
```

主要就是在过滤链接时加上了另一个条件' wssd_content. jsp? bookid in a[href]，这是经过分析得到的图书信息链接具有的特征。保存代码为 all_a_booklist. py，并运行，结果为

```
Prezi 演示进阶之路 /module/goods/wssd_content. jsp? bookid = 46449
网络安全实验教程 /module/goods/wssd_content. jsp? bookid = 46439
从零进阶！数据分析的统计基础(第 2 版) /module/goods/wssd_content. jsp? bookid = 46443
AutoCAD 2016 中文版快捷命令权威授权版 /module/goods/wssd_content. jsp? bookid = 46420
胸有成竹:数据分析的 SPSS 和 SAS EG 进阶(第 2 版) /module/goods/wssd_content. jsp? bookid =
46412
超越医疗: HIT 的拓荒——“军字一号”点滴回望 /module/goods/wssd _ content. jsp? bookid
= 46413
电商逆袭 移动端电商设计(全彩)(含 DVD 光盘 1 张) /module/goods/wssd_content. jsp? bookid
= 46403
如虎添翼:数据处理的 SPSS 和 SAS EG 实现(第 2 版) /module/goods/wssd_content. jsp? bookid =
46405
```

```
从技术走向管理——李元芳履职记(第 2 版) /module/goods/wssd_content. jsp? bookid = 46407
分布式视频编码算法与系统 /module/goods/wssd_content. jsp? bookid = 46401
```

结果正是该页图书列表包含的 10 个链接信息。可以使用很多其他的方法达到相同的目标，如可以使用 attrs = {属性名:属性值}参数，利用具有一定特征的属性值将所需的元素过滤出来，通常这类属性可以是“id”“class”，也可以是“style”等，是在进行数据抽取时常用的方法。在该实例中，希望得到的 a 元素没有“id”“class”等属性。下面给出另一种方法，就是通过逐步深入的层级定位，逐步达到希望得到的元素位置，这需要通过分析获得。比如在本例中，要获取的每个链接都位于一个 div 中，并且 div 具有属性

```
< div style = " width:730px; float:left; display:inline; margin - top:20px;margin - left:20px; border
- bottom:dotted 1px #878787;padding - bottom:20px;" >
```

因此可以定位到每个 div 元素，从中获取需要的链接。下面的代码实现了这个功能，即

```
from bs4 import BeautifulSoup
with open( beautifulSoup_set. html' , r ,encoding = utf8 ) as f:
    bs = BeautifulSoup( f. read( ) )
    div_lst = bs. find_all( div' ,attrs = { style' :
                                          ' width:730px;'
                                          ' float:left;'
                                          ' display:inline;'
                                          ' margin - top:20px';
                                          ' margin - left:20px;'
                                          ' border - bottom:dotted 1px #878787';
                                          ' padding - bottom:20px'; } )
    for div in div_lst:
        a_lst = div. find_all( a' )
        for a in a_lst:
            print( a. text. strip( ) ,a[ href' ] )
```

在 find_all 函数中的 attrs 可以限定希望找到的元素集的共有属性，这里的' style' 属性和对应的值就是上面列出的 div 中 style 属性的和值的改写，写成了字典的键和值的形式。将代码保存为 all_a_div. py 中并运行，结果为

```
Prezi 演示进阶之路 /module/goods/wssd_content. jsp? bookid = 46449
```

```
网络安全实验教程 /module/goods/wssd_content.jsp? bookid=46439
从零进阶! 数据分析的统计基础(第 2 版) /module/goods/wssd_content.jsp? bookid=46443
AutoCAD 2016 中文版快捷命令权威授权版 /module/goods/wssd_content.jsp? bookid=46420
胸有成竹:数据分析的 SPSS 和 SAS EG 进阶(第 2 版) /module/goods/wssd_content.jsp? bookid=46412
超越医疗:HIT 的拓荒——"军字一号"点滴回望 /module/goods/wssd_content.jsp? bookid=46413
电商逆袭 移动端电商设计(全彩)(含 DVD 光盘 1 张) /module/goods/wssd_content.jsp? bookid=46403
如虎添翼:数据处理的 SPSS 和 SAS EG 实现(第 2 版) /module/goods/wssd_content.jsp? bookid=46405
从技术走向管理——李元芳履职记(第 2 版) /module/goods/wssd_content.jsp? bookid=46407
分布式视频编码算法与系统 /module/goods/wssd_content.jsp? bookid=46401
```

上面的 attrs 参数中在指定键值对时，值也可以是正则表达式对象。下面的代码中就使用正则表达式来表示属性值的特征，即

```
from bs4 import BeautifulSoup
import re
with open('beautifulSoup_set.html',r,encoding='utf8') as f:
    bs=BeautifulSoup(f.read())
    div_lst=bs.find_all('div',attrs={'style':
                                     re.compile('width:730px.*')})
    for div in div_lst:
        a_lst=div.find_all('a')
        for a in a_lst:
            if a.text!='':
                print(a.text.strip(),a['href'])
```

在使用正则表达式之前，需要引入 re 模块。在 style 属性值的位置是一个正则表达式对象。该对象的模式为

```
width:730px.*
```

表示匹配以 width:730px 开头，后跟任意字符串的字符串。只要 style 满足这种匹配的字符串，均可以被 find_all 找到，在本例中，这样就可以满足要求了，在其他情况下，可能需要设计更为复杂的正则表达式应对特定的需求。将上面的代码保存在 all_a_div_2.py 并运行，结果为

Prezi 演示进阶之路 /module/goods/wssd_content. jsp? bookid = 46449
网络安全实验教程 /module/goods/wssd_content. jsp? bookid = 46439
从零进阶！数据分析的统计基础(第 2 版) /module/goods/wssd_content. jsp? bookid = 46443
AutoCAD 2016 中文版快捷命令权威授权版 /module/goods/wssd_content. jsp? bookid = 46420
胸有成竹:数据分析的 SPSS 和 SAS EG 进阶(第 2 版) /module/goods/wssd_content. jsp? bookid = 46412
超越医疗: HIT 的拓荒——“军字一号”点滴回望 /module/goods/wssd _ content. jsp? bookid = 46413
电商逆袭 移动端电商设计(全彩)(含 DVD 光盘 1 张) /module/goods/wssd_content. jsp? bookid = 46403
如虎添翼:数据处理的 SPSS 和 SAS EG 实现(第 2 版) /module/goods/wssd_content. jsp? bookid = 46405
从技术走向管理——李元芳履职记(第 2 版) /module/goods/wssd_content. jsp? bookid = 46407
分布式视频编码算法与系统 /module/goods/wssd_content. jsp? bookid = 46401

这与前面直接使用字符串作为 style 属性值的方法结果是相同的。

2.4 json 结构

json 格式的文档内容也是在数据处理中经常会遇到的。Python 提供了 json 模块用于 json 格式内容的处理。可以使用 help(json) 来查看 json 的帮助文档，然后进入 DESCRIPTION 部分，即可看到大量的实例代码，如图 2-30 所示。

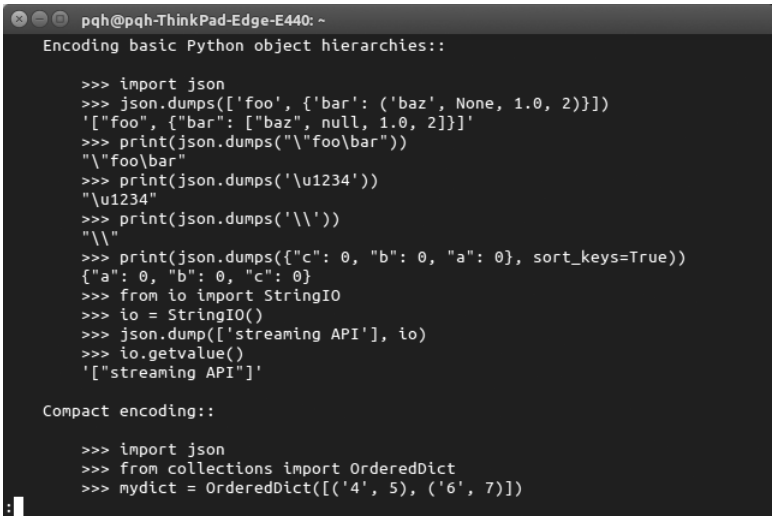


图 2-30　json 帮助文档

json 的使用比较简单，常用的方法有 load、loads、dump 及 dumps。它们是成对出现的，掌握起来并不困难。本节主要介绍如何在处理 json 数据时与 MongoDB 配合起来使用，在这个过程中会给出一些方法的使用方式。

结合 MongoDB 的好处在于提供了类 SQL 的查询，可以保存数据。面对有些复杂的网站，特别是目前的地图类网站，返回的数据格式多是 json 类型，字段很多，且嵌套层次较深。这时可以将抓取的结果直接插入 MongoDB，可自动按字段进行存储。使用 MongoDB 的客户端打开后，可以像类似 Excel 的方式查看内容，有助于进一步确定所关注的字段和字段层次，以及数据的整体形式。下面通过一个地图抓取的实例给出 json 的使用方式，以及将抓取结果存入 MongoDB 的过程。

很多提供地图服务的网站一般返回的数据都是 json 格式的数据。下面以高德地图的 json 地图数据为例进行说明。高德地图功能强大，文档丰富，官方提供了大量获取信息数据的 API，可以使用不同的 API 获得不同的信息数据。这些数据基本上都是 json 格式的，本例中主要演示 json 解析和 MongoDB 的配合使用，分析在地图单击相应位置返回的 json 数据如何保存在 MongoDB 数据库中。

使用 Firefox 浏览器，按 F12 打开 web 控制台，进入网站。以哈尔滨为例，单击“哈尔滨江北科技馆”，将弹出该地点的相关信息，如图 2-31 所示。

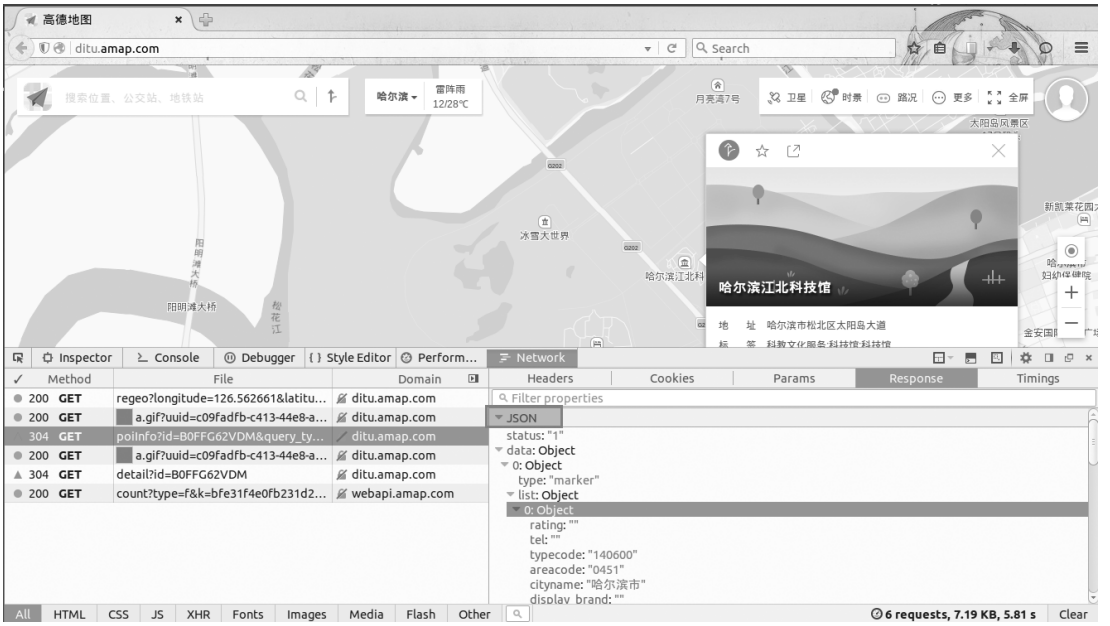


图 2-31　哈尔滨江北科技馆的相关信息

图 2-31 显示了获得信息的 url 及得到的 json 格式的返回数据。该 url 为

http://ditu.amap.com/service/poiInfo?id=B0FFG62VDM&query_type=IDQ

分析时，可以利用 requests 获得该链接的结果。requests 将在后面详细介绍。这里使用的是 requests 的 get 方法，可以模拟浏览器发出的 http 请求，并返回得到的结果对象。获得该链接结果的代码和结果如图 2-32 所示。

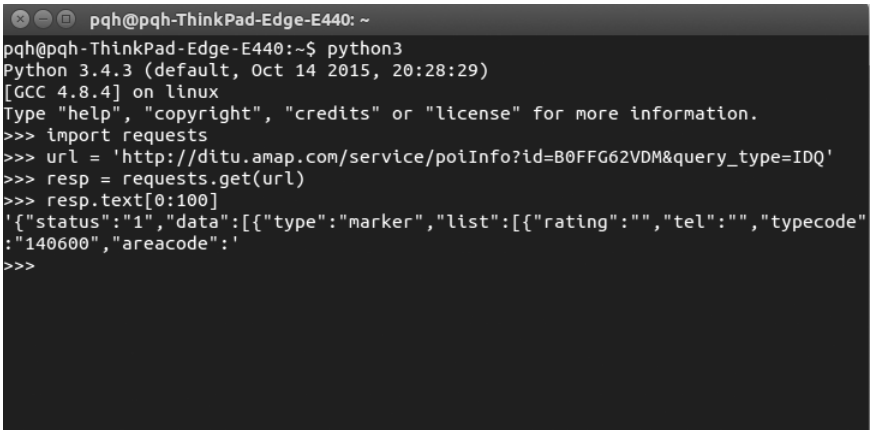


图 2-32　数据加载和结果

首先引入 requests 模块，然后将具体的链接值保存在 url 变量中，接下来就可以利用requests 的 get 方法发出请求了。请求的返回结果是一个对象，保存在 resp 变量中。resp 具有 text 属性，通过该属性可以得到相应对象的文本内容字符串。由于字符串的内容较长，因此上面的代码中对字符串进行了截取，返回了前 100 个字符串。注意，resp. text 返回的是字符串，尽管该字符串是 json 格式，但要对其分析，只能采用分析字符串的方法。json 格式在 Python 中对应的是字典类型，为了对 json 字符串信息进行方便和有效的信息提取，最好将其转为字典类型，Python 中提供了 json 模块来完成这个工作。json 模块有很多方法。其中，loads 方法可以满足现在的需求。该方法可以将一个符合 json 格式的字符串转换为字典类型的数据，如图 2-33 所示。

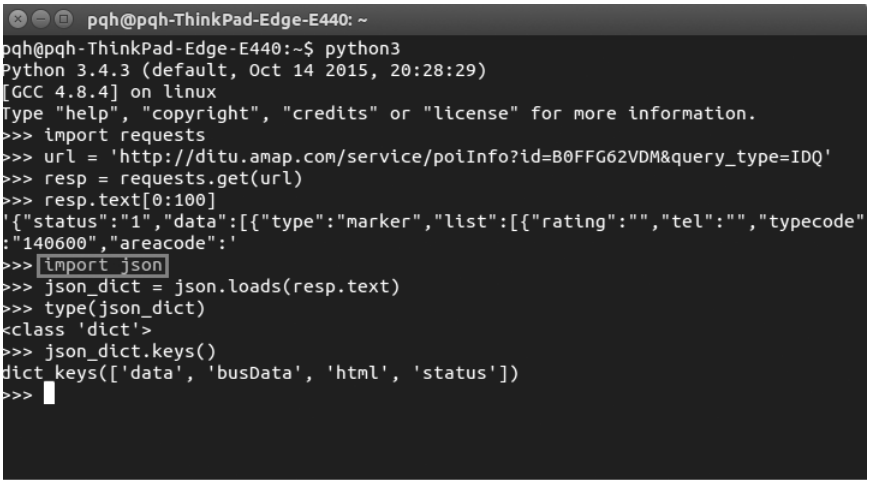


图 2-33　json 模块的引入和使用

利用 import 引入 json 模块，然后调用 json 的 loads 方法，将返回的文本作为方法的参数传入。转换的结果是与 json 字符串对应的字典，type(json_dict)的返回值 <class' dict' > 可以看出返回值为字典类型。json_dict 是一个字典，可以调用与字典相关的方法，如 keys 方法。可以看到，该字典共有 4 个键，即 data、busData、html 和 status。以 data 为例，图 2-34 是在如图 2-33 所示命令行下继续输入 json_dict[' data']的结果。

```
>>> json_dict['data']
[{'type': 'marker', 'list': [{'distance': '0', 'group_flag': '0', 'cinemazuo_fla
g': '0', 'templateData': {'isoverbooked': False}, 'longitude': '126.576445', 'lo
cation': {'lng': 126.57644500000004, 'lat': 45.776713}, 'discount_flag': '0', 'd
iner_flag': '0', 'recommend_flag': '0', 'areacode': '0451', 'rating': '', 'subPo
is': [], 'address': '太阳岛大道', 'deepinfoIcon': '0', 'id': 'B0FFG62VDM', 'mark
erType': 'marker-single', 'shape_region': '', 'exits': [], 'contain': ['太阳岛大
道', ''], 'tel': '', 'typeName': '', 'newtype': '140600', 'latitude': '45.776713
', 'cityname': '哈尔滨市', 'entrances': [], 'disp_name': '哈尔滨江北科技馆', 'dy
namic': {}, 'name': '哈尔滨江北科技馆', 'adcode': '230109', 'typecode': '140600'
, 'display_brand': '', 'cpdata': '', 'tType': '3'}]}]
>>>
```

图 2-34　json_dict[' data']的结果

这里面就是存放具体信息的位置。从返回值的形式可以看出，返回了一个列表，列表中又嵌有字典，尽管可以读出所需信息，如经纬度信息

```
' location' : { lng' : 126. 57644500000004,' lat' : 45. 776713}
```

但是想从 json_dict[' data']中一级一级地利用列表或字典的操作方法得到该信息，那么抽取的方式也不是非常明显的。可以使用一些 json 格式的解析工具将这样的信息展开，然后根据展开的层级定位到信息进行抽取。下面介绍使用 MongoDB 的方式。MongoDB 作为一种非关系型数据库，特别适用于存储这类 json 格式的信息，而且 MongoDB 提供了丰富的调用方法供未来数据提取和分析使用。此外，除了 MongoDB 的客户端，还有很多第三方提供的基于图形界面的客户端环境，环境中就包含了图形化的 json 展示界面。

下面介绍将抓取信息保存至 MongoDB 的操作步骤。首先安装 MongoDB，命令为

```
sudo apt - get install mongodb - server
```

安装后会自动启动，可以通过 ps aux | grep mongod 查看，如图 2-35 所示。

```
pqh@pqh-ThinkPad-Edge-E440:~$ ps aux|grep mongod
mongodb   6485  1.1  0.3 382060 48492 ?        Ssl  10:28   0:02 /usr/bin/mongod
 --config /etc/mongodb.conf
pqh       6688  0.0  0.0  15960  2168 pts/29   S+   10:32   0:00 grep --color=au
to mongod
```

图 2-35　通过 ps aux | grep mongod 查看

MongoDB 可以在启动服务时指定数据库存放的路径和指定的服务端口。这些参数可以在启动命令时在命令行中利用参数指定。因此，如果需要对 mongod 服务进行定制化的启

动，那么需要先关闭当前的 MongoDB 服务，然后启动新的服务，并在命令行中指定参数。

指定数据库存放位置的选项是 --dbpath，指定端口的选项是 --port，如将数据库保存在/home/pqh/mongodb/db 下，并指定端口为 10001，那么使用的命令应为

```
mongod --dbpath /home/pqh/mongodb/db --port 10001
```

注意，如果没有/home/pqh/mongodb/db 目录，那么需要先创建一个。使用的命令为

```
mkdir -p /home/pqh/mongodb/db
```

注意，命令中需要有选项 -p，表示可以创建多级目录。

图 2-36 给出了上述过程，包括创建目录、停止当前服务、启动新的服务。

```
pqh@pqh-ThinkPad-Edge-E440:~$ mkdir -p /home/pqh/mongodb/db
pqh@pqh-ThinkPad-Edge-E440:~$ ls -l  /home/pqh/mongodb/db
总用量 0
pqh@pqh-ThinkPad-Edge-E440:~$ sudo service mongodb stop
[sudo] password for pqh:
mongodb stop/waiting
pqh@pqh-ThinkPad-Edge-E440:~$ mongod  --dbpath /home/pqh/mongodb/db --port 10001
Fri Jun  3 12:26:44.905 [initandlisten] MongoDB starting : pid=4455 port=10001 d
bpath=/home/pqh/mongodb/db 64-bit host=pqh-ThinkPad-Edge-E440
Fri Jun  3 12:26:44.905 [initandlisten] db version v2.4.9
Fri Jun  3 12:26:44.905 [initandlisten] git version: nogitversion
Fri Jun  3 12:26:44.905 [initandlisten] build info: Linux orlo 3.2.0-58-generic
#88-Ubuntu SMP Tue Dec 3 17:37:58 UTC 2013 x86_64 BOOST_LIB_VERSION=1_54
Fri Jun  3 12:26:44.905 [initandlisten] allocator: tcmalloc
Fri Jun  3 12:26:44.905 [initandlisten] options: { dbpath: "/home/pqh/mongodb/db
", port: 10001 }
Fri Jun  3 12:26:44.979 [initandlisten] journal dir=/home/pqh/mongodb/db/journal
Fri Jun  3 12:26:44.979 [initandlisten] recover : no journal files present, no r
ecovery needed
Fri Jun  3 12:26:47.962 [initandlisten] preallocateIsFaster=true 31.64
Fri Jun  3 12:26:50.796 [initandlisten] preallocateIsFaster=true 33
Fri Jun  3 12:26:56.519 [initandlisten] preallocateIsFaster=true 41.58
Fri Jun  3 12:26:56.519 [initandlisten] preallocateIsFaster check took 11.539 se
cs
```

图 2-36　mongod 启动命令实例

启动成功，同时输出 MongoDB 的版本和日志信息的存放目录。

下面就应该编写 Python 代码进行抓取数据的插入了。在 Python3 中，与 MongoDB 链接需要 pymongo 模块，首先需要安装这个模块。命令为

```
sudo pip3 install pymongo
```

图 2-37 说明安装成功了。

dir(pymongo)列出了 pymongo 模块包含的各种属性和方法。有了 pymongo 模块，就可利用 Python 和 MongoDB 数据库进行交互了。下面给出代码实例，代码主要完成的功能是将地图信息 json 数据转换为 Python 字典存入到 MongoDB 中。代码如下，文件名为 json_cralwer. py。

图 2-37　pymongo 模块的引入

```
import requests
import json
import pymongo

ip = '127.0.0.1'
port = 10001
db_name = 'gd_map'
collection_name = 'pos_info'

#哈尔滨江北科技馆
url_1 = 'http://ditu.amap.com/service/poiInfo?' \
        'id=B0FFG62VDM&query_type=IDQ'
#哈尔滨太阳岛风景区
url_2 = 'http://ditu.amap.com/service/poiInfo?' \
        'id=B01C30003A&query_type=IDQ'

urls = [url_1, url_2]

mongo_conn = pymongo.MongoClient(ip, port)

db = mongo_conn[db_name]
```

```
collection = db[ collection_name]

for url in urls:
    try:
        print(url)
        resp = requests. get(url)
        json_dict = json. loads(resp. text)
        collection. save(json_dict)
    except Exception as e:
        print(e)
```

代码中首先引入了相应的模块。其中，requests 主要用于抓取数据使用；json 如前例所示，主要用于将返回的 json 字符串数据转换为 Python 的字典类型数据。pymongo 主要负责与 MongoDB 服务器进行交互。

接下来设置一些需要用到的参数，IP 设置为 127. 0. 0. 1，因为本例中 MongoDB 是在本机启动，故将 IP 设为本机地址即可。如果需要连接运行于其他机器的 MongoDB，则需要改变相应的 IP 地址。port 指定了连接 MongoDB 时所需的端口号，前面的实例中使用 10001 作为 MongoDB 服务的端口号，所以此处指定 port 为 10001，是整数 10001，而不是“10001”字符串。

下面的两个量 db_name 和 collection_name 分别给出了连接成功后，所操作的 MongoDB 数据库和集合名称。尽管不是关系型数据库，但 MongoDB 中也使用“数据库”这个名称表示与关系数据库中“数据库”相似的概念。在 MongoDB 中 collection 即集合，相当于关系型数据库中的 table（表），而 collection 中存储的元素 document（文档）相当于关系型数据库中的 row（行）。在本例中，db_name 指定为 gd_map' ，collection_ name 指定为 pos_info' ，未来数据就会插入到 MongoDB 的 gd_map 数据库的 pos_info 集合中。

本例中给出了两个抓取对象的链接，存放在 urls 列表中。下面的一句

```
mongo_conn = pymongo. MongoClient(ip, port)
```

调用了 pymongo 的 MongoClient 方法，传入设置好的 ip 和 port 作为参数，如果本地的 mongodb 已经启动，那么该句执行时就能正确地连接，并将连接对象保存到 mongo_conn 供后续代码使用。如果连接失败，那么主要的可能原因是 mongodb 服务器没有正确启动，利用前面介绍的命令

```
mongod  -- dbpath /home/pqh/mongodb/db  -- port 10001
```

启动即可。接下来的两句

```
db = mongo_conn[ db_name ]
collection = db[ collection_name ]
```

其中，第一句根据数据库名称 db_name 连接到相应的数据库，第二句再根据集合的名称 collection_name 在连接到的数据库中找到相应的集合。第一次使用时可能没有相应的数据库和集合，MongoDB 会自动创建。接下来 for 循环就是遍历 urls 列表中的链接，对于每个链接获得链接的响应数据，将数据转换为字典结构，使用集合的 sava 方法

```
collection. save( json_dict)
```

就可以将转换后的字典结构数据作为文档插入到相应的集合中了。

下面运行的这段代码执行起来比较简单，在 shell 中进入脚本 json_cralwer. py 的目录，运行 python3 json_cralwer. py 即可，运行时会插入相应的数据，程序运行结束后，与服务器建立的相关连接也会自动关闭。执行后，在 MongoDB 窗口中可以看到相应的信息，如图 2-38 所示。

```
Sat Jun  4 17:07:44.376 [FileAllocator] allocating new datafile /home/pqh/mongod
b/db/gd_map.ns, filling with zeroes...
Sat Jun  4 17:07:44.376 [FileAllocator] creating directory /home/pqh/mongodb/db/
_tmp
Sat Jun  4 17:07:44.413 [FileAllocator] done allocating datafile /home/pqh/mongo
db/db/gd_map.ns, size: 16MB,  took 0.022 secs
Sat Jun  4 17:07:44.413 [FileAllocator] allocating new datafile /home/pqh/mongod
b/db/gd_map.0, filling with zeroes...
Sat Jun  4 17:07:44.435 [FileAllocator] done allocating datafile /home/pqh/mongo
db/db/gd_map.0, size: 64MB,  took 0.022 secs
Sat Jun  4 17:07:44.436 [FileAllocator] allocating new datafile /home/pqh/mongod
b/db/gd_map.1, filling with zeroes...
Sat Jun  4 17:07:44.437 [conn3] build index gd_map.pos_info { _id: 1 }
Sat Jun  4 17:07:44.437 [conn3] build index done.  scanned 0 total records. 0 se
cs
Sat Jun  4 17:07:44.536 [FileAllocator] done allocating datafile /home/pqh/mongo
db/db/gd_map.1, size: 128MB,  took 0.099 secs
Sat Jun  4 17:07:44.898 [conn3] end connection 127.0.0.1:45086 (2 connections no
w open)
Sat Jun  4 17:07:44.898 [conn2] end connection 127.0.0.1:45080 (2 connections no
w open)
Sat Jun  4 17:08:05.247 [initandlisten] connection accepted from 127.0.0.1:45110
 #4 (2 connections now open)
```

图 2-38　MongodDB 窗口中的信息

接下来需要考虑的问题是，数据是否插入成功及存储的形式。MongoDB 自带的客户端程序 mongo 可以完成这个工作。图 2-39 给出了这个过程。

其中，

```
mongo 127. 0. 0. 1 :10001
```

用来连接 MongoDB 服务器，服务器的 IP 为 127. 0. 0. 1，端口号为 10001，这与本地所启动

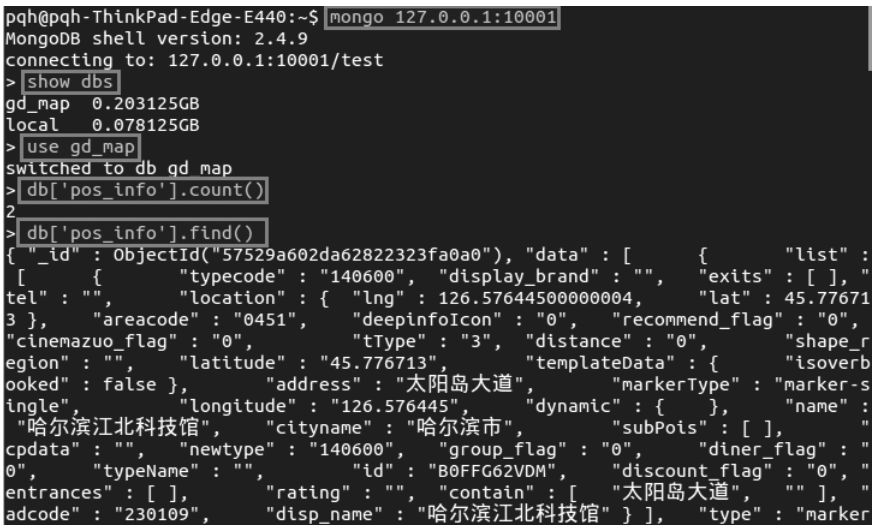


图 2-39 MongoDB mongo 客户端的使用

服务器的情况相同。下一句

```
show dbs
```

给出当前服务器上的数据库列表。根据显示结果可以看到，当前服务器上有两个数据库：一个是刚刚创建并写入的 gd_map；另一个是 local。因为要对 gd_map 进行操作，因此使用命令

```
use gd_map
```

切换到 gd_map 数据库，切换后，可以使用 db['pos_info']定位到集合“pos_info”，然后就可以调用集合支持的方法进行相关操作了。这里使用了两个方法 count()和 find()。其中，count()方法将返回集合中所含文档的个数，因为插入了两条，所以结果为 2，即

```
>db['pos_info'].count()
2
```

find()方法返回集合中包含的所有文档，也可以向 find()方法传递参数，返回根据参数形式过滤后的内容，具体的用法可以查找 MongoDB 的官方文档。

至此，通过上述一系列操作，已经可以将网上得到的 json 形式的数据进行改造并存入 MongoDB 数据了，而且还可以在客户端进行相应的操作，查看插入的情况。基于 mongo 的命令行，客户端可以执行各类操作，如进行数据的检索、修改、增加、删除等操作。除此之外，还有很多第三方提供的图形化 mongo 客户端工具，图形化的客户端工具带来了更好的数据展示效果和操作的效率。下面给出 Ubuntu 下使用 Robomongodb 客户端的方法。

进入官方网站 https://robomongo. org，如图 2-40 所示。

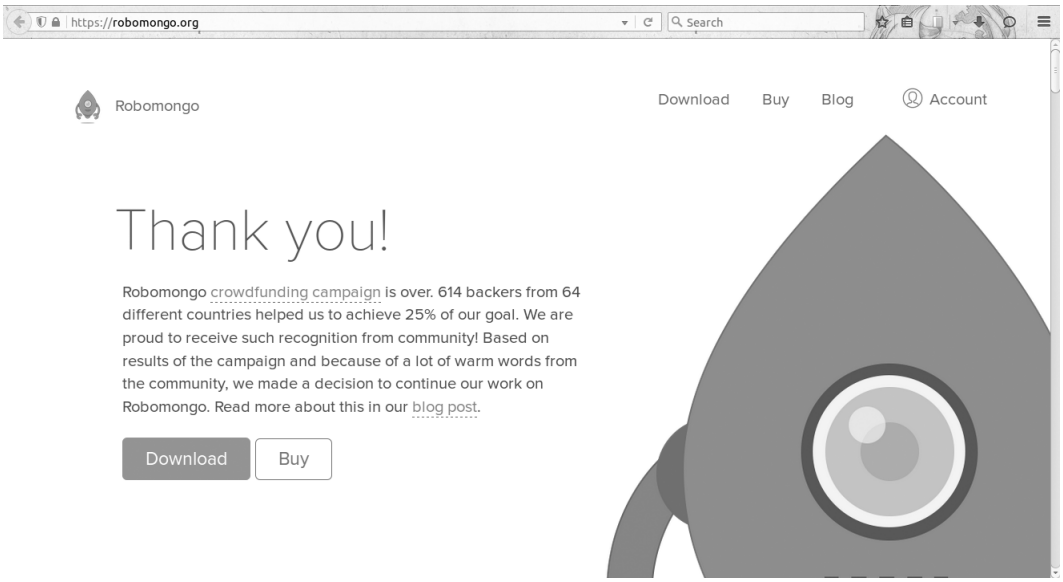


图 2-40　Robomongodb 客户端首页

在网页上单击“Download”，进入下载页，如图 2-41 所示。

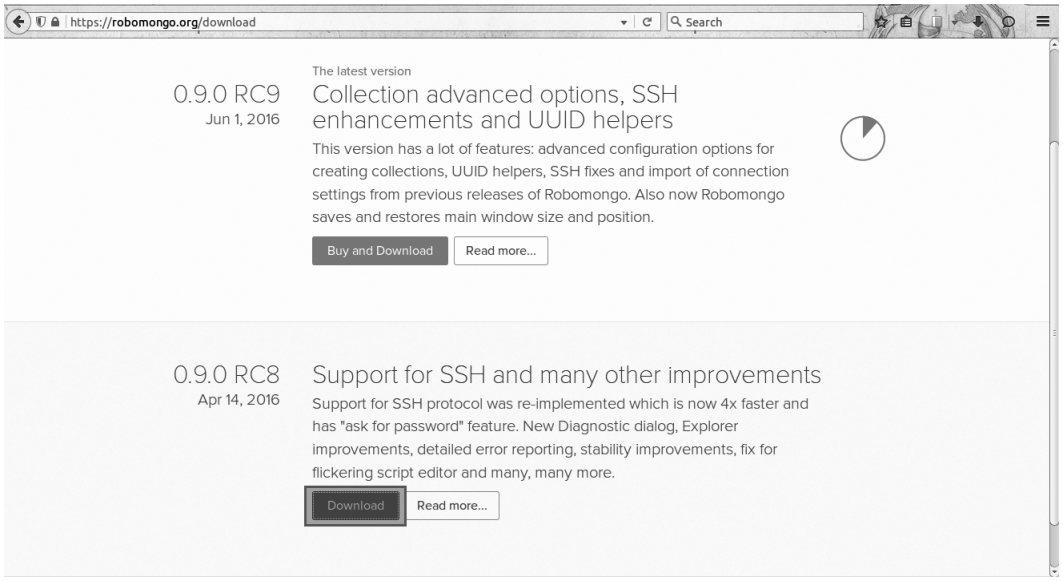


图 2-41　进入下载页

在该下载页中可以选择如图 2-41 方框所示的免费下载，单击后的界面如图 2-42 所示。

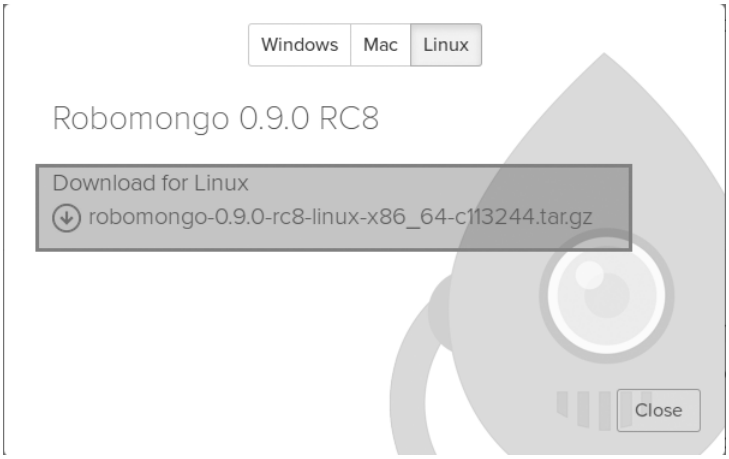


图 2-42　Linux 版本免费下载

在图中选择“Linux”，单击方框中的链接下载即可。下载后保存在相应的目录中，可以单击鼠标右键利用菜单项进行解压，如图 2-43 所示。也可以使用 tar 命令解压，使用 shell 进入压缩文件目录，然后运行 tar xvf robomongo - 0.9.0 - rc8 - linux - x86_ 64 - c113244. tar. gz 进行解压，如图 2-44 所示。

图 2-43　右键解压文件

进入解压后的 bin 目录，可以双击 robomongo 文件执行程序，也可在 shell 中在当前目录中使用命令 ./robomongo 执行程序，也可以创建链接（类似 Windows 的快捷方式），通过链接来执行。执行后会弹出连接窗口，如图 2-45 所示。

在该窗口中单击“Connect”，弹出连接设置窗口如图 2-46 所示。设置 MongoDB 服务的 IP 地址和端口号，因为是本地，所以“Address”为 localhost，后面的端口填写 10001。这是

```
pqh@pqh-ThinkPad-Edge-E440:~/programming_kits/robo$ tar xvf robomongo-0.9.0-rc8-
linux-x86_64-c113244.tar.gz
robomongo-0.9.0-rc8-linux-x86_64-c113244/DESCRIPTION
robomongo-0.9.0-rc8-linux-x86_64-c113244/bin/
robomongo-0.9.0-rc8-linux-x86_64-c113244/bin/qt.conf
robomongo-0.9.0-rc8-linux-x86_64-c113244/bin/robomongo
robomongo-0.9.0-rc8-linux-x86_64-c113244/COPYRIGHT
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/libQt5Core.prl
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/libQt5XcbQpa.so.5
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/libQt5Widgets.la
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/libQt5DBus.so
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/libQt5PrintSupport.prl
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/libQt5PrintSupport.so.5
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/libQt5DBus.la
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/platformthemes/
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/platformthemes/libqgtk2.so
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/libQt5PrintSupport.so
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/libQt5Widgets.so.5.5.1
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/imageformats/
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/imageformats/libqico.so
robomongo-0.9.0-rc8-linux-x86_64-c113244/lib/imageformats/libqgif.so
```

图 2-44　使用 tar 解压文件

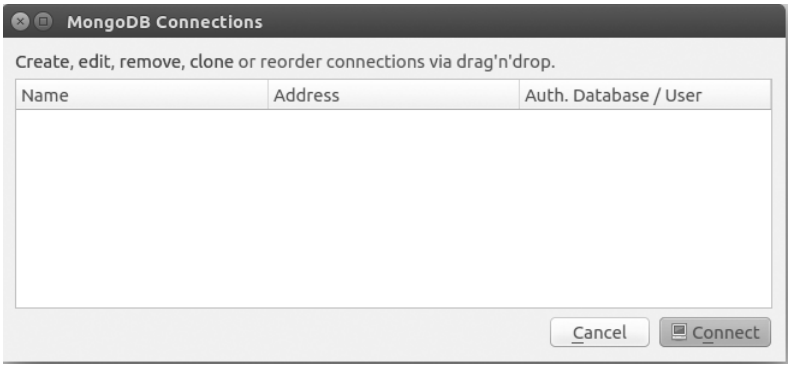


图 2-45　MongoDB 连接窗口

本例中 MongoDB 服务的指定端口号。填写后，单击“Save”回到连接界面，选中刚才编辑的连接，再单击连接就可以连接到 MongoDB 服务了。

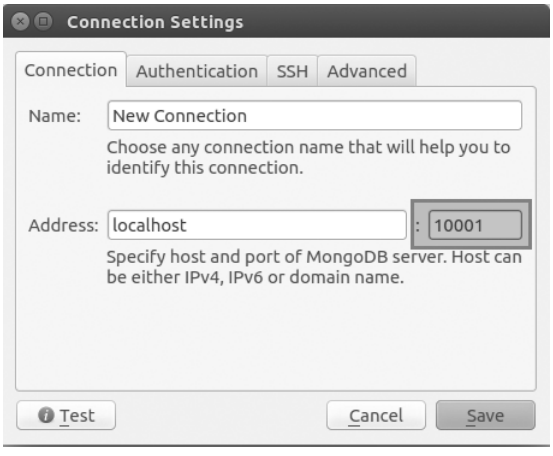


图 2-46　MongoDB 的地址和端口号

连接后的界面如图 2-47 所示。

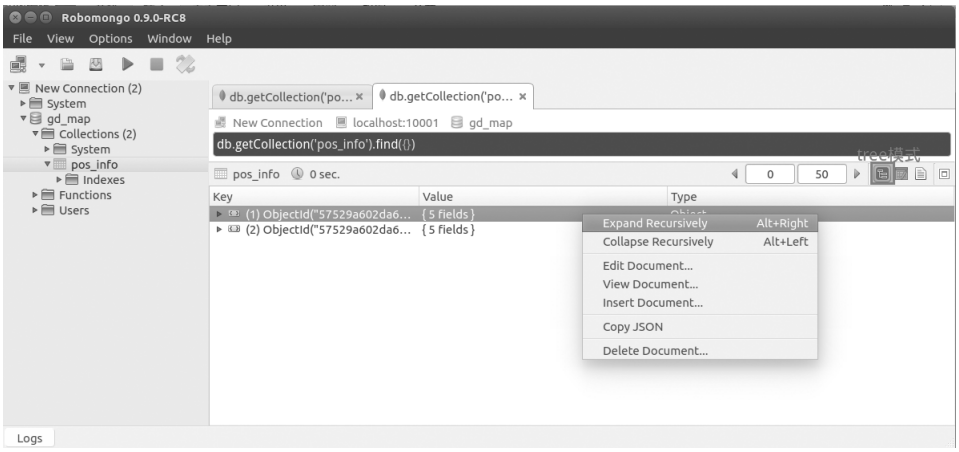


图 2-47　连接后的界面

在图的左侧导航部分可以看到建立的数据库 gd_map，在数据库的集合中可以看到建立的集合 pos_info，双击这个集合，在右侧会显示该集合的文档信息。因为插入了两条数据，所以显示了两条。默认显示的是 tree 模式的视图，右键单击某个文档，在菜单中选择“Expand Recursively”，即递归展开，就可以看到文档内部的结构了，如图 2-48 所示。

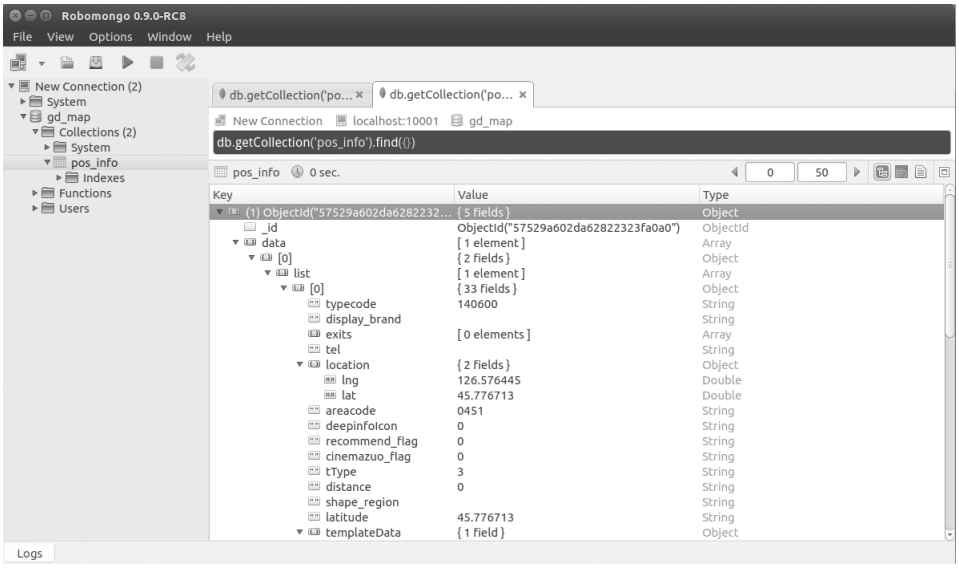


图 2-48　tree 模式下查看文档结构

图中，这种层次结构的文档展示特别有助于抽取元素时进行元素位置的分析，一目了然，比起命令行的客户端更为清晰。在 tree 模式视图旁边的按钮可以将集合展示方式变为 table 模式，如图 2-49 所示。

在 table 模式中可以看到 json 格式文档最顶层的格式，如在该 table 视图中，每个文档

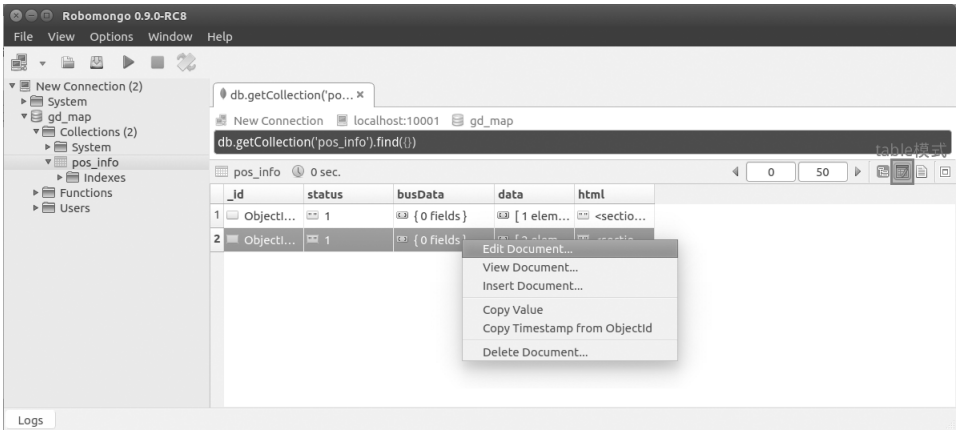


图 2-49　table 模式

都包含 5 个域。其中，_id 是插入数据时 mongdb 服务自动添加的索引，后面的 4 个域就是前面已经看到的 4 个域，如图 2-50 所示。

```
>>> json_dict.keys()
dict_keys(['data', 'busData', 'html', 'status'])
```

图 2-50　table 模式中 4 个表头域

定位到某个文档，右键单击，在弹出的菜单中选择“View Document”，可以看到文档中的具体信息，如图 2-51 所示。

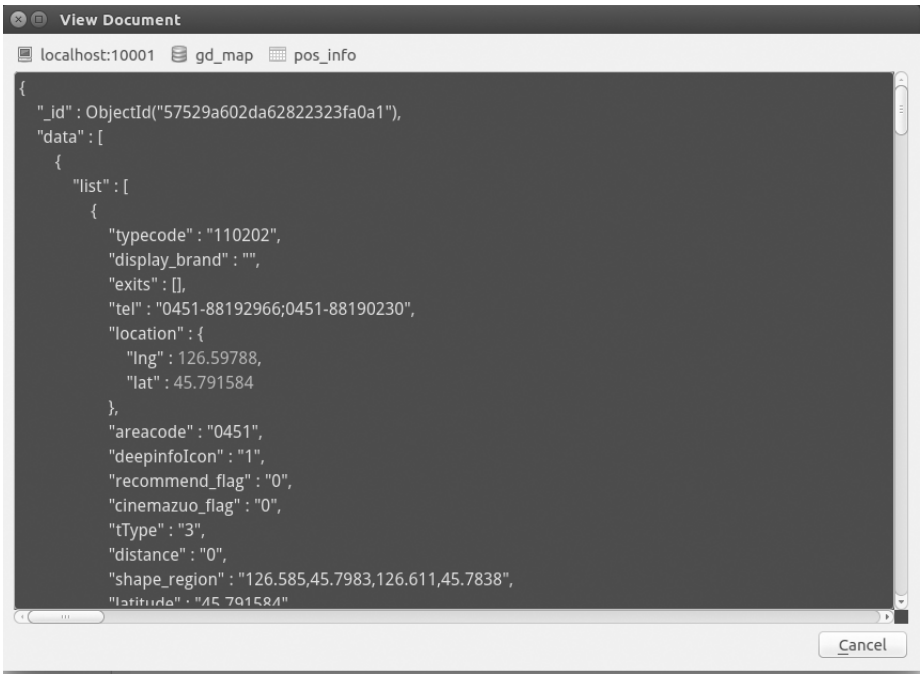


图 2-51　“View Document”内容窗口

在这种模式下可以看到文档的内容为文本格式，内容已经按照层次进行了缩进，可以很清晰地分析 json 格式文档的层次关系。

实战

查阅 Python 的 MongoDB 操作模块的方法，将存入 MongoDB 中的信息读取出来，在终端输出。

3 第3章 单机数据抓取

本章主要介绍单机数据抓取。与第4章分布式数据抓取相比，单机数据抓取开发速度快，部署容易，修改方便。本章主要讲解单机单进程（单线程）顺序数据抓取和单机下的并发与并行抓取。

对于单机单进程顺序抓取，3.1节以Spynner为例进行了介绍，当然，使用requests等其他的模块也可以实验。使用Spynner的原因在于：一方面，在抓取时可以弹出一个浏览器，在浏览器中观察抓取过程的变化，如翻页、单击链接跳转时的现象；另一方面，Spynner可以加载javascript生成的动态内容，从而保证可以抓取动态内容，这是requests有时无法做到的。在第1章举例时给出了requests的基本使用方法，在那些实例中就是requests在单进程中使用的情形，本章3.2节将给出requests更为详细的使用方法。虽然都是单机单进程抓取，但Spynner和requests给出了两种不同风格的实现形式，所以将两者同时进行介绍。在第5章将会介绍与Spynner形式类似但更为强大和灵活的Selenium。在不涉及动态内容加载时，requests更加轻量，本章后半部分的并发与并行抓取均采用requests作为抓取模块。

3.1 单机顺序抓取

单进程数据抓取是最常见的数据抓取形式，通常应用于目标明确及抓取内容数量不是很大的情况下。本节主要介绍如何利用Python进行单进程抓取，使用的库是Spynner。Spynner目前支持Python2.7，经过适当改动，将Spynner安装到Python3上。本节的代码是使用Python 3执行的，如果使用Python2.7执行，则可能需在文本显示和存储时进行适当的转码。使用Spynner需要安装Qt4，对于Pyhon2.7，可以使用sudo pip insall spynner安装Spynner。

Spynner是基于pyqt的一种库，封装了pyqt强大的webkit，具有执行javascript的能力，使用Spynner可以完全模拟一个浏览器的功能和行为。

使用 Spynner 时与其他 Python 模块的使用方法类似，都需要引入模块。

```
import spynner
```

引入后就可以使用了。Spynner 提供了大量的方法供调用，也可以称这些方法为 API。这些方法极大地方便了抓取工作。

本节以 Spynner 为例，对电子工业出版社的计算机类图书进行抓取。抓取的主要内容如图 3-1 所示。

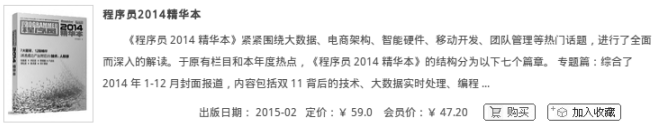


图 3-1 抓取的主要内容

抓取的主要内容包括书籍的名称、简介、出版日期、定价、会员价等。抓取的结果将存放在 excel 中。

Spynner 作为 Python 模块，使用的方式和其他的模块类似。首先，引入模块

```
import spynner
```

引入模块后，就可以调用模块提供的资源完成任务了。首先，生成一个“浏览器”，用来完成未来的抓取任务，即

```
browser = spynner. Browser( )
```

尽管只用了一句代码，但此时已经产生了一个具备基本功能的浏览器 browser 了，生成的 browser 对象提供了各种方法。这些方法在下面的使用中会一一介绍。先看一下 show( ) 方法。该方法主要用来显示 browser，通过调用这个方法就可以在抓取的过程中动态地观察页面的变化了。

```
browser. show( )
```

当然，如果在抓取的过程中不希望浏览器显示，可以不调用 show( ) 方法，此时只是不显示，但并不会影响 browser 的功能。

下面介绍关键的方法 load。load 方法接受的参数主要有：

url：要打开的目标网站链接；

load_timeout：网页加载超时设定，默认值为 10 秒；

wait_callback：回调函数，默认为 None，即无回调函数；

tries：尝试次数，默认为 None，当设为 True 时，表示无限地尝试连接，设置为整数时表示尝试连接的次数。比如，当 tries 为 4 时，load_timeout 没有设置，在默认为 10 秒的情况下，若网页无法加载，则可以以 load_timeout 为时间间隔尝试 4 次连接，若连不上，则认为连接超时。

例如，最简单的打开电子工业出版社网站的语句为

```
browser. load( http://www. phei. com. cn/ )
```

此时一个浏览器会弹出，可以认为就是刚才初始化的 browser，如图 3-2 所示。

图 3-2　Spynner 弹出的浏览器加载了网页

打开并正确加载网页后，browser 的 load 方法任务就完成了。因此，加载完毕后，load 方法退出，弹出的浏览器也会随即关闭，为了使浏览器能够多停留一会儿以便观察现象，可以利用 wait 方法。该方法接受一个时间参数，如下面的调用

```
browser. wait(10)
```

会等待 10 秒的时间，一般抓取时并不需要调用这个方法，这里只是演示使用。

需要注意的是，最后需要关闭 browser 以便彻底释放资源，相应的方法为 close。

```
browser. close( )
```

全部的代码为

```
import spynner
browser = spynner. Browser( )
browser. show( )
browser. load( http://www. phei. com. cn' )
browser. wait(10)
browser. close( )
```

在相应的目录下启动 shell，再使用 python 运行脚本时即可观察到上面的运行结果。

上面的代码只是显示了页面，通常我们对网页的信息更感兴趣。下面我们主要讨论如何抓取相应的信息。前面已经确定了需要得到的信息，也就是抓取的计划：抓取电子工业出版社所有“计算机”类图书的信息。该抓取大致分为三个步骤：

第一步，获得所有图书的链接；

第二步，根据每个图书的链接获得相应的内容，并提取出感兴趣的内容；

第三步，获得全部图书信息，将获得的信息整理，以某种方式存储。

下面实现这几个步骤。

首先，分析有关图书的链接。在网站首页的搜索框位置输入“计算机”，如图 3-3 所示，然后单击“搜索”。

图 3-3　搜索“计算机”相关图书

搜索后得到的页面为常见的分页列表形式，截取该页面的底部信息，这里是我们应关注的，如图 3-4 所示。

可以看到，我们抓取的目标图书信息此时应有 848 条，在网站上共分 85 页保存。在这种常见的列表分页模式中，一般一个列表项会包括该项目详细信息的 url 链接及一些简要的信息，如图中每一本图书的“出版日期”“定价”“会员价”等就属于这种简单的信息，而详细的信息通常需要单击链接后才能展示。这种网站结构是目前常用的分类商品信息展示方式，因此通过本节介绍的方法，其他的类似网站也可以通过相似的方式解决。

现在第一步的目标更为具体明确了，为了获得所有图书的信息，我们应该在这 85 个分页中收集所有的图书链接地址。具体来说，应该做的工作是遍历所有分页，每个分页中解析出该页包含的图书链接，形成所有链接的一个汇总列表。可见，我们关注的主要问题是“分页”和“图书链接”的解析。首先研究分页。我们回到这个页面，单击各分页链接，

图 3-4　搜索结果

观察变化，以此来分析分页链接的变化方式。单击“2”后，浏览器会跳转至第二个关于“计算机”类图书的页面，这时浏览器地址栏中显示链接

http://www.phei.com.cn/module/goods/searchkey.jsp?Page=2&Page=2&searchKey=计算机

单击“4”后，将显示分页 4 中的信息，浏览器中的链接为

http://www.phei.com.cn/module/goods/searchkey.jsp?Page=4&Page=2&searchKey=计算机

单击“84”后，链接为

http://www.phei.com.cn/module/goods/searchkey.jsp?Page=84&Page=2&searchKey=计算机

通过这几次单击可以发现分页链接的变化规律，上面链接的差异主要在第一个 Page = 后的数字，而这个数字恰是相应分页的页号。应该注意的是第 1 页，当我们在首页的搜索框内输入“计算机”进行搜索时，被导航到的默认页就是第 1 页，此时浏览器的链接为

http://www.phei.com.cn/module/goods/searchkey.jsp

这个链接形式显然与上面链接的形式不同，但当我们用发现的规律生成第 1 页链接

http://www.phei.com.cn/module/goods/searchkey.jsp?Page=1&Page=2&searchKey=计算机

后再进行访问时，也能够跳转至正确的第 1 页。一般在处理这类问题时，通常从第 2

页向后找规律，把找到的规律再应用到第 1 页上仍然可行。由此，我们可以获得所有分页的列表。下面的代码实现了这个功能。该段代码最终作为整体抓取代码的一部分，不过也可将其保存成 py 文件后运行。代码为

```
urls = [ ]

for i in range(1, 86):
    urls. append( http://www. phei. com. cn/module/goods/searchkey. jsp? Page =
                + str(i) +
                ' &Page = 2&searchKey = 计算机 )
print(urls)
```

其中，urls 用来保存所有的分页链接，for 循环用来根据上面分析的规律生成各个链接，并将链接依次存放到 urls 中。在生成每个链接时，都利用了前面的规律，是利用 str(i)中 i 的变化体现的。

运行上面的代码，将打出生成的 urls。下面是 urls 列表生成的前几条链接，即

```
[
    ' http://www. phei. com. cn/module/goods/searchkey. jsp? Page = 1&Page = 2&searchKey = 计算机 ,
    ' http://www. phei. com. cn/module/goods/searchkey. jsp? Page = 2&Page = 2&searchKey = 计算机 ,
    ' http://www. phei. com. cn/module/goods/searchkey. jsp? Page = 3&Page = 2&searchKey = 计算机 ,
    ' http://www. phei. com. cn/module/goods/searchkey. jsp? Page = 4&Page = 2&searchKey = 计算机 ,
                                        ...
]
```

以上的代码为实现第一步做了准备，下面来看第二步，如何在每个分页内解析出每本图书的链接。比如，一本书详细信息的 url 链接就是这本书的书名，如图 3-5 所示。

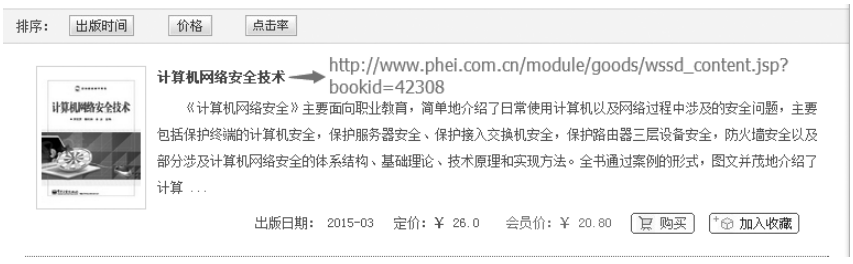


图 3-5　书名对应的详细页链接

一个分页内共有 10 本书，那么可以得到 10 个链接，85 个分页最多应该有 850 本书的链接，但当前是 848 本，这是因为最后一页只有 8 本书的信息。详细信息的 url 结构也比较

简单，如上面这本书的 url 为

```
http://www.phei.com.cn/module/goods/wssd_content.jsp?bookid=42308
```

bookid 应该是这本书的 ID 号，但我们对 ID 的数值不感兴趣，而是应该意识到这可以作为书籍链接与页面其他链接相区别的一个标识，可以用它确定一个链接是否为书籍的链接。

那么，如何从页面中分析出链接呢？其实页面内容是 HTML 文档字符串，链接也是字符串，本质上这个工作就是从大的 HTML 文档字符串中提取出我们需要的字符串。这是一种字符串处理的工作，但 HTML 文档是一种结构化文档，我们需要提取的链接信息是该文档的一个组成部分，因此可以利用这种结构化特点来进行信息提取。这方面 Python 模块有很多，如前面介绍的 BeautifulSoup 和 lxml，但这里我们还是围绕 Spynner，利用 Spynner 提供的方法解决这个问题。

下面的方式所采取的思路是：获得一个页面的全部链接信息，然后过滤出书籍的链接信息。利用刚才生成 browser 对象的 webframe 属性，然后调用 findAllElements 方法。这个方法是 QWebFrame 提供的，可以返回所要找到元素的集合。比如，我们希望返回 browser 当前页面的所有 a 元素，相应的语句为

```
a_list = browser.webframe.findAllElements('a')
```

这样，a_list 就会包含该页内出现的所有 a 对象。可以先通过 print 语句大致查看 a_list 中的元素，即

```
for a in a_list:
    print(a)
```

得到的结果为

```
<PyQt4.QtWebKit.QWebElement object at 0x00000000034AAEB8>
<PyQt4.QtWebKit.QWebElement object at 0x00000000034AADD8>
<PyQt4.QtWebKit.QWebElement object at 0x00000000034AAEB8>
<PyQt4.QtWebKit.QWebElement object at 0x00000000034AADD8>
<PyQt4.QtWebKit.QWebElement object at 0x00000000034AAEB8>
...
```

可以看出，a_list 中的每个元素都是一个对象，这并不是我们希望得到的，我们的目标是得到书籍链接的字符串表示，那么下一步就需要从每个对象中提取出其对应的链接，这一步骤可以用下面的实例进行说明。一本书籍的链接为

```
<a href="/module/goods/wssd_content.jsp?bookid=42345" target="_blank">计算机:一部历史</a>
```

其中，“计算机：一部历史”是该链接在网页中显示的文本信息，而我们希望得到的是 href 中包含的字符串，该字符串表示的是一个相对链接，如果在页面上单击“计算机：一部历史”就会跳转到这本书的详细信息页，因此我们需要的就是这个链接字符串。那么，如何获得这个链接呢？其实，href 是 a 的一个属性，同样 target 也是 a 的一个属性，从该角度考虑，就可以利用 QWebElement 提供的相应方法得到这个属性的值。具有这个功能的方法名为 attribute，可接受一个参数，这个参数就是某个属性，如我们希望得到链接对象 a 的 href 的值，使用的语句为

```
a.attribute('href')
```

可以对前面的 for 循环进行改变来输出每个 a 对象的 href 属性

```
for a in a_list:
    print(a.attribute('href'))
```

输出部分结果片段为

```
                    …
    javascript:gotopage('<!--firsttype-->');
    javascript:gotosort('cbsj','1','计算机');
    javascript:gotosort('jg','1','计算机');
    javascript:gotosort('djl','1','计算机');
  /module/goods/wssd_content.jsp?bookid=42345
  /module/goods/wssd_content.jsp?bookid=42345
/module/goods/shopcar0.jsp?bookid=42345&num=1
        javascript:shoucang2(42345);
            javascript:shoucang();
  /module/goods/wssd_content.jsp?bookid=42308
  /module/goods/wssd_content.jsp?bookid=42308
/module/goods/shopcar0.jsp?bookid=42308&num=1
        javascript:shoucang2(42308);
                    …
```

可以看到，使用 attribute 方法确实返回了 a 的属性值。该值为字符串类型，正好是我们需要的。但是这种方式返回了页面全部的链接，前面给出了希望得到链接的实例“/mod-

ule/goods/wssd_content. jsp? bookid = 42345”，那么可以看到，上面的部分输出中有两个链接符合要求，即

```
/module/goods/wssd_content. jsp? bookid = 42345
/module/goods/wssd_content. jsp? bookid = 42308
```

需要进行设置来筛选出希望得到的链接，希望得到的链接中有“bookid = ”标识，但像

```
/module/goods/shopcar0. jsp? bookid = 42308&num = 1
```

这种链接不是所需要的，因此过滤出所需形式链接的条件可以初步定为包含“bookid = ”同时不含“shopcar0. jsp”。之所以称为初步，是因为为了定位更精确的结果可能需要逐步地设置过滤条件，这是一个根据观察结果逐步迭代的过程。同时需要注意的是，可以有很多种方式用来过滤信息，如上面的条件可以设定为包含“bookid = ”同时不以“&num = 1”作为结尾，也可能找到所有符合条件的链接，这个条件可能需要正则表达式知识。总之，条件设定方式非常灵活，可根据实际情况和对相关知识点的掌握程度进行选择。

另外，可以观察到像“/module/goods/wssd_content. jsp? bookid = 42345”这样满足条件的链接存在两条，一般发生这种情况是因为对一本书可以使用两种方式进入详细页：一种是单击书名链接；另一种是单击书的图片。这两种方式都会跳转到相同的信息，因为图片所指向的链接和书名指向的链接是相同的，如对于上面的书名链接，其图片链接为

```
< a target = "_blank" href = "/module/goods/wssd_content. jsp? bookid = 42345" >
    < img height = "120" border = "0" width = "90" title = "计算机:一部历史"
src = "http://218. 249. 32. 138/thumbs/9787121255113. jpg" > </a >
```

可以很清楚的看到与书名的链接是相同的。这种情况也发生在其他类型的网站中，只需得到两个链接中的一个链接就可以了。下面的代码段可实现这个功能，即

```
needed_list = [ ]
for a in a_list:
    href_val = a. attribute( href )
    if 'bookid' in href_val and 'shopcar0. jsp' not in href_val:
        if href_val not in needed_list:
            needed_list. append( href_val)
```

其中，needed_list 用来存放在一个分页中需要的 href 链接；第一个 if 条件用来实现所需

书籍 href 链接部分的过滤，所使用的规则与前文描述的一致；第二个 if 条件主要是针对书名和图片 href 相同的情况，每次只将不在 needed_list 中的 href 附加进去，就可保证 needed_list 中 href 链接的唯一性。下面的代码打印出在一个分页中收集到的有效 href，即

```
for href in needed_list:
    print(href)
```

结果为

```
/module/goods/wssd_content.jsp?bookid=42345
/module/goods/wssd_content.jsp?bookid=42308
/module/goods/wssd_content.jsp?bookid=42302
/module/goods/wssd_content.jsp?bookid=42290
/module/goods/wssd_content.jsp?bookid=42257
/module/goods/wssd_content.jsp?bookid=42223
/module/goods/wssd_content.jsp?bookid=42196
/module/goods/wssd_content.jsp?bookid=42069
/module/goods/wssd_content.jsp?bookid=41747
/module/goods/wssd_content.jsp?bookid=41598
```

这与实际情况是一致的，一个分页中的图书列表中包含 10 本书的信息。

到此，有两种方式可以选择。一种是根据链接分别进入每本图书的详细信息页获得详细信息，然后进入下一个链接指向的详细信息页获得信息，依此类推，直到这个分页的全部链接处理完毕。之后，处理下一个分页，获得里面所有图书的链接，依次获得详细信息，直到所有的分页和链接处理完毕。这种方式将获得链接和获得详细信息两个过程耦合到了一起，如果在抓取的过程中网络中断，或者在处理抽取信息时遇到了不规则的信息格式导致异常，都会干扰抓取的进行，尽管在下次抓取时可以从上次中断的位置开始，但这需要在中间存储一些文本日志信息，使处理逻辑变得复杂。另一种方式是根据上面得到一个分页全部所需 href 链接的方法获得所有分页的链接信息，然后分别处理这些链接，得到所需的详细信息，是前面列出的三个步骤所采取的方案。这种方式的好处是，将获得链接和获得信息的步骤完全独立，每一步保留该步骤的执行结果，使每一步的目标更加明确，最大程度地降低各种异常情况对抓取的影响。

根据这个思路，我们首先完成前面提到的第一步：获得所有图书的链接。

基于上面获得一个分页内 href 列表的方法，这个工作比较容易完成。下面的代码可完成这个工作，即

```
urls = [ ]

for i in range(1, 86):
    urls. append( http://www. phei. com. cn/module/goods/searchkey. jsp? Page =
                + str(i) +
                ' &Page = 2&searchKey = 计算机 )
    print(urls)
    all_href_list = [ ]
    for url in urls:
        browser. load(url, load_timeout = 60)
        a_list = browser. webframe. findAllElements( a' )
        needed_list = [ ]
        for a in a_list:
            href_val = a. attribute( href' )
            title = a. toPlainText( )
            if' bookid' in href_val and' shopcar0. jsp' not in href_val and title ! =' :
                if [title, href_val] not in needed_list:
                    needed_list. append([title, href_val])
        all_href_list + = needed_list
```

为了连贯，代码片段中又给出了 urls 的获得方式。all_href_list 的作用是存放所有分页的 href 链接信息；对于 for 循环中的 url，将遍历 urls 列表中存放的分页地址；对于每个 url，browser. load(url, load_timeout = 60)将加载这个页面，load_timeout 的值表示页面加载超时的时间值，这个值的默认值是 10，即 10 秒，这里设为 60，也可以设为更大的值。接下来的代码基本功能前面已经介绍过了，这里做了一点改动。其中，

```
title = a. toPlainText( )
```

表示获得 a 的文字部分，如 < a href = “www. xxx. com” > 123 < /a > 。其中，123 就是 a 的文字部分，也就是页面上所显示的可供单击的文字链接。之所以需要得到文字部分，是因为一般文字和链接是一一对应的，这样做的一个好处是通过链接找到详细的信息，然后可以轻松地将文字和详细信息关联起来，形成一条完成的信息记录。同时要注意的是，对于下面这种形式

```
< a target = "_blank" href = "/module/goods/wssd_content. jsp? bookid = 42345" >
    < img height = "120" border = "0" width = "90" title = "计算机:一部历史
        "src = "http://218. 249. 32. 138/thumbs/9787121255113. jpg" > < /a >
```

a. toPlainText( )的“结果为空串”，因为 < a … > … 间不是文字，而是 < img > 标识的图片，所以得到“空串”的结果。为了避免这种情况，在 if 中增加了 title ！='' ；的判断。现在 needed_list 中的每个元素不只包含 href_val，还应包含与其对应的 title，可以将这两者组织成列表[title，href_val]加入 needed_list 中，此时 needed_list 将是列表的列表。if [title，href_val] not in needed_list 可以不再需要，但这里保留了，为的是更清晰地表达逻辑关系。最后一句 all_href_list + = needed_list 表示将 needed_list 的结果合并到总的链接列表 all_href_list 中，可以将 all_href_list 的内容写到一个文本文件中，作为第一步执行完毕得到的结果，即

```
all_href_file = open( all_hrefs. txt' ,' w' )

for href in all_href_list:
    all_href_file. write( \t' . join(href) +' \n' )

all_href_file. close( )
```

文件 all_hrefs. txt 将保存 all_href_list 中的内容,' \t' . join(href) +' \n' 能保证每个链接在文本文件中占一行。其中,' \t' . join(href)表示将 href 表示的[title，href_val]列表用 \t' 连接起来，形成一个以\t 为分隔符的字符串，而 \n' 表示保存该串时在行尾加一个换行，使每个字符串在保存的文件中占一行。all_hrefs. txt 文件的部分内容如图 3-6 所示。

all_hrefs.txt ×

```
1  计算机：一部历史  /module/goods/wssd_content.jsp?bookid=42345
2  计算机网络安全技术 /module/goods/wssd_content.jsp?bookid=42308
3  计算机应用基础习题集（Windows XP+Office 2007）（修订版） /module/goods/wssd_content.jsp?bookid=42302
4  计算机应用基础（职业模块）(Windows 7+Office 2010)（第2版）（含CD光盘1张） /module/goods/wssd_content.jsp?bookid=42290
5  用微课学·计算机应用基础（双色） /module/goods/wssd_content.jsp?bookid=42257
6  计算机控制系统（第2版）/module/goods/wssd_content.jsp?bookid=42223
7  计算机网络基础    /module/goods/wssd_content.jsp?bookid=42196
8  大学计算机基础    /module/goods/wssd_content.jsp?bookid=42069
9  大学计算机组成原理教程（第2版） /module/goods/wssd_content.jsp?bookid=41747
10 计算机主板维修不是事儿(含DVD光盘1张)/module/goods/wssd_content.jsp?bookid=41598
11 计算机图形学（第四版） /module/goods/wssd_content.jsp?bookid=41577
12 计算机网络与网络计算  /module/goods/wssd_content.jsp?bookid=41563
13 计算机视觉特征提取与图像处理（第三版）   /module/goods/wssd_content.jsp?bookid=41486
14 计算机硬盘维修与数据恢复高手    /module/goods/wssd_content.jsp?bookid=41455
```

图 3-6　all_hrefs. txt 文件的部分内容

至此，第一步的任务完成了。下一步的工作是根据 all_href. txt 中的内容得到每本书的详细信息。首先打开一本书的详细信息页面，如“计算机：一部历史”这本书，链接为 http：//www. phei. com. cn/module/goods/wssd_content. jsp? bookid = 42302，详细信息如图 3-7 所示。

下面的任务是获得位于图片右侧的所有信息，这些信息的格式类似于 Python 数据结构字典中的元素形式，即键值对，这对我们来说是一个启发，也可以按这种键值的方式保存数据信息。另外，还要取得内容简介下的文字，这些组合起来就可以作为较为完整的数据

图 3-7　图书详细信息

记录了。下面我们来完成这个任务，主要使用前面介绍的技术及 Python 自身方便的字符串处理方法。

首先，获得图片右侧的类似于键值对的信息。浏览器呈现的是经过渲染的 HTML 代码，而我们获取的信息正是来源于 HTML 代码本身，因此在希望得到信息前，首先需要了解信息在 HTML 代码中的表示形式。当前，各种主流浏览器自身具备的调试功能为我们提供了方便。下面以 FireFox 浏览器为例说明这个过程。使用浏览器打开上面的链接后，单击 F12 调出调试器。FireFox 还有一款强大灵活的插件 Firebug，安装后也可以按照上面的方式调出。这里使用的是 Firebug 2.0.8。如果没有 Firebug，则使用 FireFox 自身的调试器也可以。当然，使用 Chrome 或 IE 提供的调试工具也可以，使用方式都是类似的。

页面加载后，弹出调试器后的浏览器界面如图 3-8 所示。

然后，按如图 3-9 所示的标注进行操作。

操作完成后，定位到如图 3-10 所示阴影所定位的 td 元素，单击 td 元素左边的加号，即可出现矩形框内的内容。

如果继续按照这个模式进行信息单击查看操作会发现，图片右侧包含的每一条信息都存放在一个 td 项中，那么就可以使用前面获得所有 a 元素的方式获得所有 td 元素，然后找出或过滤出我们希望得到的 td 元素。对于获得所有 td 元素，则使用下面的代码即可

```
td_list = browser. webframe. findAllElements( td )
```

图 3-8　弹出调试器的浏览器页面

图 3-9　如何定位一个页面元素

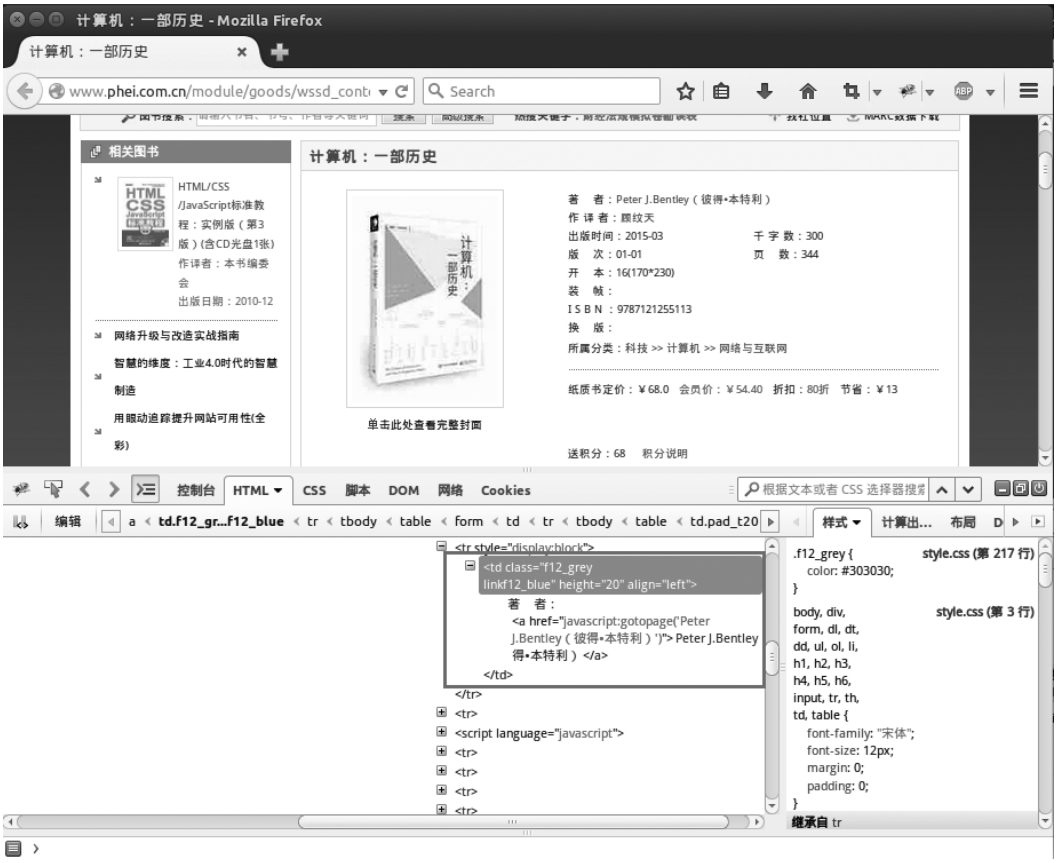


图 3-10　定位到的元素内容

那么，如何过滤呢？因为我们希望得到以冒号“：”分开的键值对表示的信息，因此可以使用这个条件作为过滤的参考。另外需要指出的是，前面给出的 toPlainText( )方法将给出元素内部的文字，对于像下面这种嵌套形式

```
<td height="20" align="left" class="f12_grey linkf12_blue">著     者：<a href="javascript:gotopage(Peter J. Bentley(彼得・本特利))">Peter J. Bentley(彼得・本特利)</a></td>
```

也就是 td 中嵌套 a 的形式，当对外层 td 元素使用 toPlainText( )方法时，a 的文字 Peter J. Bentley（彼得・本特利）也会被提取出来，下面的代码给出了这个过程，即

```
import spynner

browser = spynner.Browser()
browser.show()
```

```
browser.load('http://www.phei.com.cn/module/goods/wssd_content.jsp?bookid=42345',
             load_timeout=60, tries=3)
td_list = browser.webframe.findAllElements('td')

for td in td_list:
    td_text = td.toPlainText()
    if ':' in td_text:
        print(td_text)

browser.close()
```

将代码保存为任意名称以 .py 为后缀的文件，运行的结果片段如图 3-11 所示。

```
……
定　价：￥59.0
会员价：￥47.0
嵌入式Linux从入门到精通
定　价：￥59.8
会员价：￥47.0

定　价：￥59.8
会员价：￥47.0
  计算机：一部历史

点击此处查看完整封面
著　　者：Peter J.Bentley(彼得•本特利)
作 译 者：顾纹天
出版时间：2015-03　　千 字 数：300
版　　次：01-01　　页　　数：344
开　　本：16(170*230)
装　　帧：
I S B N ：9787121255113
换　　版：
所属分类：科技 >> 计算机 >> 网络与互联网
纸质书定价：￥68.0　会员价：￥54.40　折扣：80折　节省：￥13
送积分：68　　积分说明
……
```

图 3-11　运行的结果片段

可见，除了我们需要的信息之外，很多其他的信息也找到并输出了，这也说明网站的信息页是使用 table 布局的。前面介绍的 BeautifulSoup 可以使我们快速地定位到所需信息所在的 table 并进行信息抽取，然后在此 table 内抽取信息。在这里我们使用 Spynner 提供的 QtWebKit 接口进行操作。其实上面的代码有一些问题，问题在于对于 td.toPlainText()，如果 td 的子孙元素中包含 table 元素，并且 table 中还有 td，那么根据 toPlainText()方法的特点，前面 td 的输出内容一定包含子孙层次的 td 中包含的文字，图中是一个输出片段，如果全部显示，就可以看出这个问题，因此还需进一步调整来获得我们想要的信息。这时仍可借鉴第一步中对 a 的处理，a 的 href 属性的值就是希望得到的链接值，有时元素的属性可以轻松地将元素与其他元素分开。此处我们如何区分出我们希望从中得到信息的 td 元素呢？td 元素的属性不止一个，经过比较，利用 height 属性可以达到这个效果，我们想得到信息

的 td 的 height 为 20，利用这个观察可以使用如下的代码实现过滤，即

```
for td in td_list:
    if td.attribute('height') == '20':
        txt = td.toPlainText()
        print(txt)
```

使用这段代码置换上面代码段中相应的位置，运行后直接得到如图 3-12 所示的结果。

```
丛书名 :
著　　者：Peter J.Bentley(彼得•本特利)
作 译 者：顾纹天
出版时间：2015-03
千 字 数：300
版　　次：01-01
页　　数：344
印刷时间：
开　　本：16(170*230)
印　　次：01-01
装　　帧：
I S B N ：9787121255113

重　　印：新书
换　　版：
广告语：
纸质书定价：￥68.0　　会员价：￥54.40　　折扣：80折　　节省：￥13

送积分：68　　积分说明

计算机：一部历史
与此 件组合商品一同购买
总定价：￥
组合价：￥
```

图 3-12　根据 td 的 height 过滤结果

此时基本上满足我们的需要了。假设我们只关心方框中的内容，那么可以将书的全部信息模仿键值对的形式放到一个字典中，然后在需要时根据提供的键得到相应的值。下面的代码实现了这个功能。其中，语句 if ':' in txt: 的主要目的是绕过图中的空行，txt.split(':')的作用是将 txt 文字以':'作为分隔符分为两部分，得到的是一个列表，如 A:B.split(':')将得到['A','B']。代码中关于 k、v 的赋值语句可分别得到键 k 和值 v，字典 d 用来保存这些键值对。

```
d = {}
for td in td_list:
    if td.attribute('height') == '20':
        txt = td.toPlainText()
        if ':' in txt:
            print(txt)
```

```
            k = txt.split(':')[0]
            v = txt.split(':')[1]
            d[k] = v
print(d)
```

运行得到 d 的内容为

```
{'计算机':'一部历史','印刷时间':'','印\xa0\xa0\xa0\xa0次':'01-01','著\xa0\xa0\xa0\xa0者':'Peter J. Bentley(彼得·本特利)','版\xa0\xa0\xa0\xa0次':'01-01','换\xa0\xa0\xa0\xa0版':'','丛书名':'','ISBN':'9787121255113','千\xa0字\xa0数':'300','作\xa0译\xa0者':'顾纹天','出版时间':'2015-03','总定价':'¥','开\xa0\xa0\xa0\xa0本':'16(170*230)','送积分':'68\xa0\xa0\xa0\xa0积分说明','装\xa0\xa0\xa0\xa0帧':'','页\xa0\xa0\xa0\xa0数':'344','广告语':'\xa0\xa0\xa0','纸质书定价':'¥68.0\xa0\xa0\xa0\xa0会员价','组合价':'¥','重\xa0\xa0\xa0\xa0印':'新书'}
```

其中出现了很多\xa0 符号，因此可以在上面的代码段中将语句

```
txt = td.toPlainText()
```

改为

```
txt = td.toPlainText().replace('\xa0','')
```

再次运行，得到 d 为

```
{'纸质书定价':'¥68.0 会员价','组合价':'¥','著者':'Peter J. Bentley(彼得·本特利)','送积分':'68 积分说明','作译者':'顾纹天','总定价':'¥','丛书名':'','千字数':'300','ISBN':'9787121255113','广告语':'','印刷时间':'','页数':'344','计算机':'一部历史','出版时间':'2015-03','版次':'01-01','印次':'01-01','开本':'16(170*230)','换版':'','重印':'新书','装帧':''}
```

这时已经将无关的符号替换掉了。注意，Python 中的字典是无序的，因此在两次运行的结果中，信息显示的顺序并不一样。

图 3-13 中的内容简介是另一部分要获取的内容。

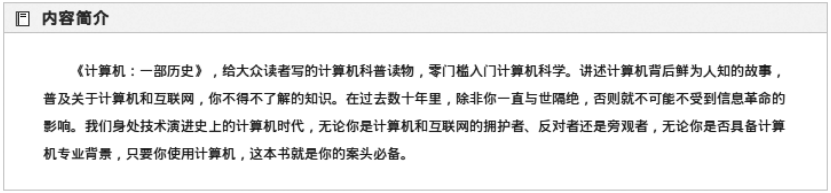


图 3-13　内容简介

这时仍可以使用上面介绍过的分析 HTML 代码利用 dom 元素定位所需内容的方式，但有时也可以使用更为灵活的方法——字符串切割的方式，即前面使用过的 split 方法，split 根据提供的分隔符对字符串进行分割，分割的结果以列表返回，如

```
http://www.baidu.com/
```

如果以 / 作为分隔符进行分割，代码为 'http://www.baidu.com'.split('/')，结果为

```
['http:','','www.baidu.com','']
```

此时被切割的字符串就根据提供的分隔符被分割开了，由于返回的结果是一个列表，所以可以利用列表索引的方式提取出被分割的部分，如

```
['http:','','www.baidu.com','',][2]
```

将得到 'www.baidu.com'，如果希望得到 'baidu'，则可以继续使用 '.' 进行分割，然后利用索引得到

```
'www.baidu.com'.split('.')[1]
```

将得到 'baidu'，如果将上面的过程写成一条语句，就是下面常见的字符串分割方式，即

```
'http://www.baidu.com/'.split('/')[2].split('.')[1]
```

通常在使用 split 时对分隔符适当选取，可以有助于我们快速地定位所需信息，如果先以 '.' 作为分隔符，那么得到 'baidu' 的语句为

```
'http://www.baidu.com/'.split('.')[1]
```

这样在效率上要高一些。此外，需要注意分隔符不一定是单个字符，字符串也可以作为分隔符，如

```
'http://www.baidu.com/'.split('http://')
```

将得到['','www.baidu.com/']。

可以说，对一个页面反复使用 split 操作总可以得到我们希望得到的信息，但因为对一次抓取同一网站的各所需页面都具有类似的 HTML 文档结构和 dom 元素定位，因此使用各种通过解析文档结构来定位和获取所需信息的方式更为直接和有效。相比之下，split 方式显得繁琐和复杂些，不过在有些情况下使用 split 会弥补基于元素定位方式的不足。最常见的情况是，有时网页看似结构整齐，但所需信息在不同信息页上的定位并不一致，如果没

有特定的属性信息（class，id，href，color…）来进行定位。就需要使用 xpath 进行定位。显然，信息位置不同，xpath 也不相同。xpath 定位所需元素的方法也走不通，这时就要使用 split 了。一般来说，使用 split 可以与各种基于元素定位的方式结合使用，基本上任何信息都可以准确地抽取到。

回到上面的实例，如何利用 split 获得简介内容呢？仍以上面的“计算机：一部历史”为例，可以打开页面的 HTML 源码文件找到内容简介在 HTML 中的位置，在 < td align = "left" valign = "top" class = "line_h24 f12_grey pad_t20 pad_bot20" > < p > 和 < /p > 之间，而前者是页面中独有的，页面中还有一处 < td align = "left" valign = "top" class = "line_h24 f12_grey pad_t20 pad_bot20" >，但没有链接 < p >，因此这种独有性正好可作为 split 的分割标识。可定义函数来完成这个功能，函数为

```
def get_instruction(html):
    l_tag = '<td align="left" valign="top" class="line_h24 f12_grey pad_t20 pad_bot20"><p>'
    r_tag = '</p>'
    instruction = ''
    try:
        instruction = html.split(l_tag)[1].split(r_tag)[0]
    except Exception as e:
        print(e)
    return instruction
```

其中，l_tag 和 r_tag 分别代表所取信息的左右分隔标识，instruction 是要返回的书的内容简介，在使用 split 进行分割时进行了异常处理，因为如果有的页面没有左右分隔标识，那么可能会在列表索引操作时产生异常。假设 Spynner 浏览器已经打开了这本书的网页，那么可以将 browser.html 传入这个函数，得到需要的简介信息，即

```
s = get_instruction(browser.html)
print(s)
```

得到的 s 输出就是内容简介。

至此，我们得到了关于一本书的详细信息和内容简介部分，下一步就可以根据这个方式获得全部图书信息，然后将信息以某种方式存储起来。其中，获得全部信息可以通过循环来实现，每次循环依次打开一个网页处理一本书的信息，每本书的信息可以保存到一个列表中，然后加入到一个统一的全部信息列表，完成循环后，可以对这个列表进行处理。在实践中，可能存在这样的问题，就是在抓取的过程中发生了网络断线，或者程序异常终止，这时会干扰抓取过程。为了避免这种情况，一个比较可行的方法是先将要抓取信息的

网页保存到本地，待这个过程完成后，再对这个文件夹下的所有文件进行统一的信息处理，这样可以使抓取过程各阶段彼此分工更为明确，提高效率。同时，抓取并保存到本地的网页本身就是一个信息源，包含着比我们所提取的信息更多的内容，如果未来需要进一步提取或分析，则不必再从网络获取，可大大提高工作效率。

另外，如何对提取的信息进行存储也是一个重要的问题。通常这是根据实际需要确定的，可以使用 csv、tsv 等文本格式，也可以使用 Excel、关系型数据库和非关系型数据库。这里我们使用 Excel 进行信息存储，因为使用文本方式时，对于像内容简介这样的大块文字，里面可能包含了各种符号，这会与逗号和 tab 符这类分隔符相混淆，同时以文本形式保存时，一行的显示区域有限，可能会造成信息查看不便、干扰信息的理解。除了这里使用的 Excel 保存方式外，也可以采用前面介绍的基于 MongoDB 的保存方案，此外也可以选择其他的关系型数据库，如 Mysql、Oracle、Sqlite 等。这些数据库的使用和操作方式类似，都需要安装相应的 Python 数据库模块。前面已经安装和使用了 MongoDB 的 pymongo 模块，读者可以按照这种方式下载其他数据库模块进行安装和使用，操作相应的数据库。

下面解决上面提出的问题。首先是得到所有需要的网页 HTML 文件，将它们保存在本地目录。所有图书的名称和详细信息页的部分 url 在前面已经被保存在 all_href. txt 中了，为了讨论方便，下面再次给出文件的部分截图，如图 3-14 所示。

all_hrefs.txt ×

```
1   计算机：一部历史  /module/goods/wssd_content.jsp?bookid=42345
2   计算机网络安全技术 /module/goods/wssd_content.jsp?bookid=42308
3   计算机应用基础习题集（Windows XP+Office 2007）（修订版） /module/goods/wssd_content.jsp?bookid=42302
4   计算机应用基础（职业模块）(Windows 7+Office 2010)（第2版）（含CD光盘1张） /module/goods/wssd_content.jsp?bookid=42290
5   用微课学·计算机应用基础（双色） /module/goods/wssd_content.jsp?bookid=42257
6   计算机控制系统（第2版）/module/goods/wssd_content.jsp?bookid=42223
7   计算机网络基础    /module/goods/wssd_content.jsp?bookid=42196
8   大学计算机基础    /module/goods/wssd_content.jsp?bookid=42069
9   大学计算机组成原理教程（第2版） /module/goods/wssd_content.jsp?bookid=41747
10  计算机主板维修不是事儿(含DVD光盘1张)/module/goods/wssd_content.jsp?bookid=41598
11  计算机图形学（第四版） /module/goods/wssd_content.jsp?bookid=41577
12  计算机网络与网络计算   /module/goods/wssd_content.jsp?bookid=41563
13  计算机视觉特征提取与图像处理（第三版）   /module/goods/wssd_content.jsp?bookid=41486
14  计算机硬盘维修与数据恢复高手    /module/goods/wssd_content.jsp?bookid=41455
```

图 3-14　all_href. txt 文件内容

可以读取这个文件，然后将里面的内容逐行读出。对于每一行的内容，可以分成两部分，书名和详细信息页对应的部分 url，然后基于部分 url 得到完整的 url，获得信息页后，以书名作为文件名，将网页保存在指定的目录下。假设目录名为 htmls，在当前目录下建立该文件夹。下面的程序可实现网页的获取和保存功能，即

```
import spynner
import glob
#1
browser = spynner. Browser( )
browser. show( )
```

```
#2
having_files_list = glob.glob('./htmls/*.html')
having_names_list = []
for f in having_files_list:
    having_names_list.append(f.split('./htmls/')[1].split('.html')[0])

#3
all_hrefs_f = open('all_hrefs.txt','r')
base_url = 'http://www.phei.com.cn'
#4
for line in all_hrefs_f.readlines():
    #5
    book_name = line.split('\t')[0].replace('/','_')
    book_part_url = line.split('\t')[1].rstrip()
    book_id = book_part_url.split('bookid=')[1]
    #6
    if book_name not in having_names_list:
        url = base_url + book_part_url
        browser.load(url, load_timeout=120, tries=3)
        html_f = open('./htmls/' + book_name + '_' + book_id + '.html','w')
        html_f.write(browser.html)
        html_f.close()
        print('saving:',book_name)
    else:
        print('existing!',book_name)
#7
all_hrefs_f.close()
browser.close()
```

下面再一次解释代码各部分的主要功能，按照#数字的注释顺序说明。

#1 利用 Spynner 模块初始化一个浏览器 browser，并调用 browser 的 show 方法，这样在抓取的过程中就可以动态观察网页的变化了。需要指出的是，如果不调用 show 方法，就不会出现浏览器的可视化窗口，但不会影响抓取。同时，如果开始调用 show 方法，之后将浏览器关闭或最小化，也不会影响抓取的进行。

#2 获得已经抓取到的所有网页的文件名列表。之所以加上这段代码，主要是考虑抓取可能不会一次成功，如网络中断或异常，因此在程序前加上这样一段代码，收集已经抓到

的网页名称列表，在后面每次执行抓取时，先判断要抓取的网页名是不是在这个列表中。如果在，则说明已经抓取过了，否则再执行此次抓取。当然，也可以将每次执行抓取的信息统一保存到一个文件中，在每次抓取前读入这个文件，进行一些判断，效果是一样的。

```
glob.glob('./htmls/*.html')
```

这条语句将返回./htmls/目录下所有以.html 结尾的文件名组成的列表。因为文件名中包含路径前缀./htmls/，所以使用 split 将其分割，只取需要的部分添加到 having_names_list 中。

#3 打开前面保存的 all_hrefs.txt 文件，在下面会对该文件的内容进行读取。base_url 将与每本书提供的 part_url 相拼接形成一个完整的 url。

#4 是循环主体，包含两部分#5 和#6。

#5 对于 all_hrefs.txt 的每一行进行 split 分割，分别获得书名 book_name 和部分链接 book_part_url，同时获得链接中的 book_id。这是因为我们将利用 book_name 和 book_id 一起获得文件名作为网页保存时使用的名字，之所以这样做是因为书名可能会重复，如“计算机组装与维护实用教程”这个书名，对应两本不同的书，因此结合名字与 ID 号可以解决这个问题。

#6 如果形成的书名不在已抓取书名列表，那么就要对其进行抓取和保存，同时输出'saving:' + 文件名信息。否则，提示该文件已经存在了，接着进行下一个链接的抓取。

#7 抓取结束后，关闭 all_hrefs.txt 文件，并关闭浏览器。

按照上面的代码进行一次或多次抓取，最终将获得 849 个文件，与 all_ href.txt 中包含的行数是一致的。打开 htmls 文件夹，如图 3-15 所示。

图 3-15　htmls 文件夹内容列表

下面接着利用前面介绍的获得信息的方法对每个网页文件进行分析，就可以取得每本书的信息了。这应该是最关键的一步，因为我们抓取的目的也是如此。对于上面这些文件，

因为是同一个网站的同级（详细信息级）页面，所以页面结构是相似的，而这又导致分析方法是相似的，因而可将前面定义的提取信息的方法应用于各个页面抽取相应的内容，再将所有内容一起以某种格式存储起来，这就是我们接下来要做的工作。

前面介绍的页面信息提取形式主要有两个：一个是将页面以键值对的形式表示信息，利用 Python 的字典数据结构保存起来；另一个是 get_instruction 函数，该函数的作用是提取内容简介字段。这两种方式收集到的信息一起构成了一个页面所需得到的信息。第一个方法是将页面上以键值对存储的信息仍以键值对的形式存储在字典中，但一个页面的键值对很多，未必都是我们所需要的，这时就需要给出一种方法从一个页面所有键值对形成的字典中将感兴趣的键值对的信息提取出来。假设我们最终需要的信息是关于下面列表中给出的键对应的值信息

```
key_names = [' 书名 ',' 作译者 ',' 出版时间 ',' 千字数 ',' 版次 ',' 页数 ',' 开本 ',' 内容简介 ']
```

可以定义一个函数实现这个目的，即

```
def get_val_from_dict(d,k):
    try:
        return d[k]
    except Exception as e:
        print(k,e)
        return' ----'
```

函数 get_val_from_dict 接受一个字典 d 和一个键 k，结果是返回该键对应的值 d[k]，之所以设计成函数，是因为直接使用 d[k]可能出现 d 中并没用键 k 的情况，这时取值会出现异常。这是一种常见的情况。因为对于某些网站来讲，其拥有对象的信息是不完整的，因此在页面上显示时，对于某对象因为没有某些信息，页面上就自然不会显示，也不会被抓取到，所以若用上面的方式设置好统一的所需信息列表对页面信息的提取时，会因为某些页面没有信息而触发异常，这种情况可以返回' 或用某些其他的字符串来表示没有该信息，上面函数中的 except 分支返回 ----' 就是这种情况。注意，get_val_from_dict 的逻辑使用前面介绍字典的 get 方法也可以解决，将 get 的第二个参数设为 “ ----” 即可。这里再提一下在抓取过程中将页面保存的好处。比如，我们对信息的处理是抓取一页，使用 key_name中的字段提取一页信息，将一页结果保存，那么当抓取完成时，发现漏掉了一个字段怎么办？此时只能重新抓取了，当页数庞大时将是一件很费时的重复劳动，所以保存网页到本地是较为稳妥的做法。有了对每一页信息提取的方式，接下来就是遍历前面保存的所有详细信息的页面依次提取信息，将信息汇总并存储。下面给出全部的代码，然后分别说明。

```
import spynner
import glob
import xlwt3

#1
def get_instruction(html):
    l_tag = '<td align="left" valign="top" class="line_h24 f12_grey pad_t20
pad_bot20">'
    r_tag = '</p>'
    instruction = '---'
    try:
        instruction = html.split(l_tag)[1].split(r_tag)[0]
    except Exception as e:
        print('instruction:',e)
    return instruction
def get_val_from_dict(d,k):
    try:
        return d[k]
    except Exception as e:
        print(k,e)
        return '---'

#2
browser = spynner.Browser()
browser.show()
#3
files_list = glob.glob('./htmls/*.html')
key_names = ['书名','作译者','出版时间','千字数','版次','页数','开本','内容简介']
all_recs = []
all_recs.append(key_names)

#4
for file_name in files_list:
    book_name = file_name.split('.html')[0].replace('./htmls/','')
    f = open(file_name,'r')
    html_str = f.read()
```

```
        f.close()
        #5
        browser.webview.setHtml(html_str)
        browser.wait(1)
        #6
        td_list = browser.webframe.findAllElements('td')
        d = {}
        d['书名'] = book_name
        print(file_name)
        for td in td_list:
            if td.attribute('height') == '20':
                txt = td.toPlainText().replace('\xa0','').replace('','').replace('\t','')
                if ':' in txt:
                    k = txt.split(':')[0]
                    v = txt.split(':')[1]
                    d[k] = v
        instruction = get_instruction(html_str)
        d['内容简介'] = instruction
        all_recs.append([get_val_from_dict(d,kname) for kname in key_names])

#7
wb = xlwt3.Workbook()
sheet = wb.add_sheet("图书信息")
for i in range(0,len(all_recs)):
    for j in range(0,len(all_recs[i])):
        sheet.write(i,j,all_recs[i][j])
wb.save("图书信息.xls")

browser.close()
```

#1 这两个函数的功能前面已经介绍过了。需要注意的是，get_instruction 中也进行了异常处理，主要是处理因取不到“内容简介”的内容而触发的异常。

#2 初始化浏览器。

#3 此部分主要的内容有：

glob.glob('./htmls/*.html')将返回在 htmls 目录下所有以 .html 结尾的文件的文件名组成的列表，这里的文件名是包含路径名称的。

key_names 是希望获得信息的键列表外加两个元素“书名”和“内容简介”。其中，键

的数目可以根据需要增减，这里给出的键字段都是在分析中可方便分解的，最直接的体现是在对原始 HTML 提取文字时，在文本中以单行形式表现键值对的信息，如

```
A:B
```

可以直接使用 A:B . split(':')[0]获得键 A,' A:B . split(':')[1]获得值 B。如果一行有多个键值形式信息，如

```
A:B C:D
```

就需要进一步处理了，可以综合使用 Python 的字符串处理函数进行解析。为了描述方便，key_names 中的键字段都是提取出原始 HTML 文档的文字后以一行一个键值对信息的形式存放的。对于 书名 ，因为在保存一本书的详细信息 HTML 文档时是以书名和 ID 号作为 HTML 文件名存放的，因此较易获得。对于 内容简介 使用 get_instruction 取得。

all_recs 初始化为[ ]，将作为存储从所有图书中提取出的信息列表。一本书的信息可以以列表形式存储。本例中，一本书的信息列表有 8 个字段，即

```
[info0,info1,info2,info3,info4,info5, info6,info7]
```

与 key_names 字段相对应。为了在存储时能包括表头，all_recs. append(key_names)将 key_names 中的字段作为表头先存入 all_recs 中，因此 all_recs 的最终结构为

```
all_rec = [
    [ 书名 ', 作译者 ', 出版时间 ', 千字数 ', 版次 ', 页数 ', 开本 ', 内容简介 ],
    [info00,info01,info02,info03,info04,info05, info06,info07],
    [info10,info11,info12,info13,info14,info15, info16,info17],
    ...
    [infon0,infon1,infon2,infon3,infon4,infon5, infon6,infon7]
]
```

#4 该部分为一个循环，每次循环依次从前面获得的书名列表中取出一本书进行处理。book_name 将保存获得的书名。下面的 3 句代码用来保存打开的文件，取得文件内容存放在 html_str 中。

#5 文件内容加载。此部分应该指出的是，在获得 HTML 内容字符串后，可以有各种方式进行解析，如 BeautifulSoup 或 lxml 来提取出其中的信息，但此处主要是给出使用 Spynner 进行抓取的实例，所以采用的方式都是围绕 Spynner 的。Spynner 是 QtWebkit 的进一步封装，但 Spynner 并没有将所有有关 QtWebKit 的方法提供方法调用接口，不过可以利用 Spynner 已提供的接口作为桥梁来进一步探索 QtWebKit。

比如，我们需要将页面加载到浏览器，然后利用设计的信息提取方法进行抽取。那么，如何将已有页面加载至浏览器中呢？使用 Spynner 自身的 load 方法是做不到的，但是利用 QtWebKit 中 webview 的 setHtml 方法可以做到这点，代码为

```
browser. webview. setHtml(html_str)
```

此时浏览器加载的 html_str，browser. html 就是 html_str，与从网络上打开链接并加载网页内容的效果是一样的。下一句

```
browser. wait(1)
```

的作用是让浏览器等待 1 秒，这是根据经验设定的，如果不用此句，则会造成第 1 页的内容有时不能完全加载。

#6 该部分主要用于信息的提取，具体的功能和每本书的信息组织方式前面已经介绍过了。

#7 将得到的信息保存到 Excel 中。Python3 使用 xlwt3 进行 Excel 写操作。代码中的双重 for 循环，i 代表行，j 代表列，而需要保存的内容——all_rec 在前面已经介绍过了，所以此处不难理解。

运行代码完毕后，打开当前文件夹下的图书信息 . xls，就可以看到我们收集到的信息了，如图 3-16 所示。

	A	B	C	D	E	F	G	H
1	书名	作译者	出版时间	千字数	版次	页数	开本	内容简介
2	全国计算机等级考试真考题库、高	全国计算	2013-04	549	01-01	196	16(210*	本书依据教育部考试中心最新发布的《全国计算机等级考试考试
3	计算机应用基础实训手册（第2版）	李京宁	2011-08	474	01-01	296	16(185*	本书是"计算机应用基础"课程的实训手册，可与各种版本的《计
4	计算机组装、维护与维修_27771	匡松	2010-08	430	01-01	272	16(185*	本书共15章，主要内容包括：计算机硬件基础知识、主机内部组
5	ASP.NET程序设计情境式教程_37	《全国高	2013-08	422	01-02	264	16(185*	本书以3561人才培养模式组织教材的编写，按照软件开发的工作
6	穿越计算机的迷雾_21242	李忠	2011-01	328	01-01	268	16(170*	本书以制造一台能全自动做加法的机器为主线，介绍电学、数学
7	计算机网络实用技术教程_23864	刘义常张	2008-08	358	1-01	224	16(185*	本书是一本关于计算机网络实用技术类的教材，全书共8章，第一
8	计算机组成原理_28228	罗克露等	2004-08	636	1-13	387	16(185*	本书以电子科技大学编写的六。五、八。五、九。五等全国规划
9	全国计算机等级考试专用辅导教程	希赛教育	2013-01	397	01-01	260	16(185*	本书由希赛教育等考学院组织编写，内容紧扣教育部考试中心新
10	AutoCAD计算机辅助设计模块教程	闫旭辉	2013-05	308	01-01	192	16(185*	本书以AutoCAD 2012中文版为软件基础，采用"任务驱动式"的
11	计算机应用基础_22025	肖甘	2011-09	390	01-01	244	16(185*	高职课程基于工作过程的教学是由实践来确定典型的工作任务，
12	计算机操作基础_25692	新华教育	2009-01	496	重印	392	16(185*	本书是新华电脑教育专业标准化课程系列教材中的一本，内容涉
13	计算机组装与维护情境实训（第2	雒褚建立	2014-09	506	01-01	316	16(185*	本书主要介绍如何识别、选购计算机硬件系统各配件，组装和拆
14	全国计算机等级考试专用辅导教程	希赛教育	2012-01	512	01-01	256	16(205*	本书由希赛教育等考学院组织编写，作为全国计算机等级考试三
15	全国计算机等级考试标准教程（考	严惠	2011-08	444	01-01	356	16(205*	本书紧扣最新版考试大纲，结合编者多年从事命题、阅卷及培训
16	汽车计算机控制（第2版）_20853	司利增著	2007-04	413	1-01	249	16(185*	本书是对《汽车计算机控制》第一版的全面修订，重构了体系，
17	数字信号处理——基于计算机的方	[美]Sanj	2011-10	1165	01-01	640	16(185*	本书是在数字信号领域的经典教材Digital Signal Processing:

图 3-16　Excel 文件的内容

为了说明在抓取中保存网页的重要性，我们思考这样一个场景：获得每本书的价格信息。图 3-16 中并没有包含价格信息，而作为商品来讲，价格是一个十分重要的属性信息，如果在抓取过程中没有将网页保存下来的话，此时就需要重新再抓取一次，而前面的过程已经保存了全部图书的网页信息，因此需要做的工作就只需再一次扫描全部的存储网页，得到相应的价格信息，代码的变化也不是很大。首先看如图 3-17 所示。

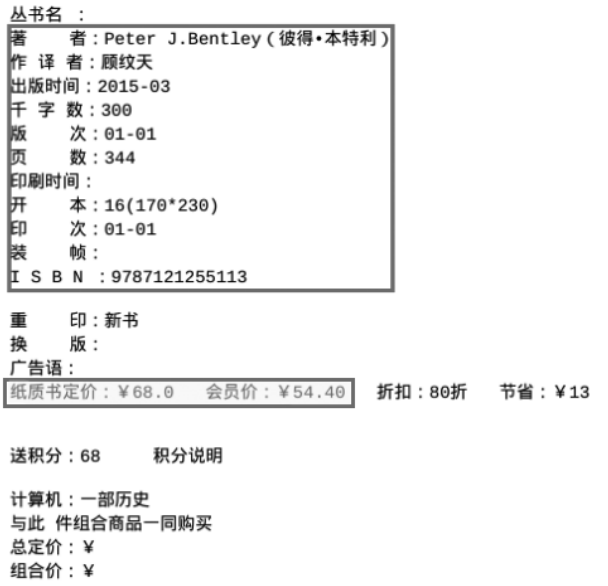


图 3-17　定价信息

其中，希望得到纸质书的定价为 68.0，而这一行中还有三个键值对类型的信息，“会员价”“折扣”和“节省”，这样就不能利用前面代码段中的

```
k = txt. split(':')[0]
v = txt. split(':')[1]
```

语句获得键和值了，因为这两句假设 txt 中只有一个以键值对形式标识的信息。此时可再次使用 split 函数来获得希望得到的信息。

打开任意一个保存的图书信息，通过查找“纸质书定价”定位，以上面的书为例定位到语句片段，

```
……纸质书定价：￥68.0   <span class="f12_red">会员价：……
```

设 s ≐ 纸质书定价：￥68.0   会员价'：，那么 68.0 可通过下面的方式得到，即

```
s. split('纸质书定价：￥')[1]. split('<span class="f12_red">会员价')[0]. replace(' 
  ', '')
```

需要注意的是，split 在划分字符串时得到的是一个列表，希望通过索引方式找到列表元素再进一步划分时，可能由于列表中没有相应的索引元素而产生异常，前面获得 k、v 的方式其实也可能存在这个问题，因此最好在分析页面时在涉及 split 操作时，利用

try. . . except 来捕获异常。下面的代码给出了获得图书价格的方式，为了保持结构完整，可再次给出获得其他信息的语句页。

```
......
key_names = [ 书名 ', 作译者 ', 出版时间 ', 千字数 ', 版次 ', 页数 ', 开本 ', 价格 ', 内容简介 ]
all_recs = [ ]
all_recs. append( key_names)

#4
for file_name in files_list:
    book_name = file_name. split( . html )[0]. replace( . /htmls/ ", )
    f = open( file_name', r )
    html_str = f. read( )
    f. close( )
    #5
    browser. webview. setHtml( html_str)
    browser. wait( 1)
    #6
    try:
        td_list = browser. webframe. findAllElements( td )
        d = { }
        d[ 书名 ] = book_name
        print( file_name)
        for td in td_list:
            if td. attribute( height ) == 20 :
                txt = td. toPlainText( ). replace( \xa0 ", ). replace( ", ). replace( \t ", )
                if ': in txt:
                    k = txt. split( ': )[0]
                    v = txt. split( ': )[1]
                    d[ k ] = v
        d[ 价格 ] = html_str. split( 纸质书定价：¥ )[1]. split( <span class = "f12_red" > 会员
价 )[0]. replace(    ", )
        instruction = get_instruction( html_str)
        d[ 内容简介 ] = instruction
        all_recs. append( [ get_val_from_dict( d,kname) for kname in key_names] )
    except Exception as e:
        print( e)
```

```
                continue
        ......
```

图 3-18 是获得的结果。

	A	B	C	D	E	F	G	H	I	J	K	L
1	书名	作译者	出版时间	千字数	版次	页数	开本	价格	内容简介			
2	全国计算	全国计算	Apr-13	549	01-01	196	16(210*285)	32.8	本书依据教育部考试中心最新发布			
3	计算机应	李京宁	2011-08	474	01-01	296	16(185*260)	34.0	本书是"计算机应用基础"课程的实			
4	计算机组	匡松	2010-08	430	01-01	272	16(185*260)	28.0	本书共15章，主要内容包括：计算			
5	ASP.NET	《全国高	2013-08	422	01-02	264	16(185*260)	33.0	本书以3561人才培养模式组织教			
6	穿越计算	李忠	2011-01	328	01-01	268	16(170*235)	36.0	本书以制造一台能全自动做加法的			
7	计算机网	刘义常张	2008-08	358	1-01	224	16(185*260)	22.0	本书是一本关于计算机网络实用技			
8	计算机组	罗克露等	2004-08	636	1-13	387	16(185*260)	30.0	本书以电子科技大学编写的六。王			
9	全国计算	希赛教育	2013-01	397	01-01	260	16(185*260)	35.8	本书由希赛教育等考学院组织编写			
10	AutoCAD	闫旭辉	2013-05	308	01-01	192	16(185*260)	29.0	本书以AutoCAD 2012中文版为			
11	计算机应	肖甘	2011-09	390	01-01	244	16(185*260)	29.0	高职课程基于工作过程的教学是由			
12	计算机操	新华教育	2009-01	496	重印	392	16(185*260)	41.0	本书是新华电脑教育专业标准化课			
13	计算机组	褚建立	2014-09	506	01-01	316	16(185*260)	40.0	本书主要介绍如何识别、选购计算			
14	全国计算	希赛教育	2012-01	512	01-01	256	16(205*260)	35.8	本书由希赛教育等考学院组织编写			
15	全国计算	严惠	2011-08	444	01-01	356	16(205*260)	38.0	本书紧扣最新版考试大纲，结合编			
16	汽车计算	司利增著	2007-04	413	1-01	249	16(185*260)	29.0	本书是对《汽车计算机控制》第一			
17	数字信号	[美]Sanj	2011-10	1165	01-01	640	16(185*235)	69.0	本书是在数字信号领域的经典教材			

图 3-18　包含价格信息的 Excel 文件

上面就是使用 Spynner 进行抓取的较为详细的流程了。应该说明的是，从形式上看，无论是使用什么模块作为抓取的主要工具，抓取的流程都是相似的，如使用 ghost 或 Selenium 等。后面将会介绍的 Selenium 也是一种可以用来抓取的利器，特别是可以方便地用于分布式数据抓取，有了 Spynner 的使用经验后，学习起来会比较简单。另外，在信息的处理中主要依靠 Python 自身的字符串函数，有时对于一些情况，利用一些封装好的模块进行HTML代码解析和信息抽取可能更便于提取页面信息。

3.2　requests

上一节使用 Spynner 给出了一个抓取的实例，主要是因为 Spynner 可以模拟一个浏览器，在抓取的时候可以动态显示整个抓取进展，同时模拟浏览器 Spynner 可以对 javascript 实现的动态内容进行抓取。后面的章节会给出更为有效的 Selenium 库可以代替 Spynner 的工作。本节介绍 requests 库。它是 Python 的 http 库，可以完成绝大部分与 http 应用相关的工作，因此可以进行数据抓取工作。与 Spynner 不同，requests 是一种非界面化的库。requests 不能模拟浏览器的全部行为，但对于一些常规的抓取工作，使用起来还是很方便的。requests 提供了丰富的且易于调用的多种方法实现对 http 相关方式的模拟。本节只围绕两个基本方法 get 和 post 的相关内容进行介绍。它们是在抓取时经常使用的方法。

1. get 方法

在向服务器发出 http 请求时，通常有 get 和 post 两种形式，而 requests 中的 get 和 post 方法可以看成是这两种形式的实现。下面的代码实现了抓取电子工业出版社网站首页的功能。

```
import requests
url = http://www.phei.com.cn/
resp = requests.get(url)
with open('home.html', 'w',encoding = 'utf8') as f:
    f.write(resp.text)
```

首先引入 requests 库，然后使用 get 方法得到系统的首页，get 的返回值 resp 是一个 Response 类的对象，对象的 text 属性包含了相应的 HTML 文本，文本被保存在脚本目录下的 home.html 中。打开后，在浏览器中的显示效果如图 3-19 所示。

图 3-19　home.html 的显示效果

response 还有其他一些属性，在交互环境下输入上面的代码可进一步考察一些属性，可以使用 help(resp)查看 response 的各种属性和方法。下面给出了 url、status_code、encoding 等属性的内容。

```
>>> import requests
>>> url = http://www.phei.com.cn/
>>> resp = requests.get(url)
>>> resp.status_code
200
>>> resp.url
'http://www.phei.com.cn/'
>>> resp.encoding
'UTF-8
```

encoding 属性显示了 resp.text 的编码，如上面显示 resp 的编码为 utf-8，resp.text 就是以此进行编码的。这个属性可以改变，改变后，使用 resp.text 访问响应内容时，会根据新的编码值进行编码。

```
>>> resp.text
```

这时，交互界面返回页面内容的最后几行如图 3-20 所示。

```
le></td>\n        <td width="189" align="center" valign="top" class="pad_t8 pad_
bot8 bod_left_grey4"><table width="100%" border="0" cellspacing="0" cellpadding=
"0">\n\t\t\t\t <!--EHM_SystemNav?pcid=106&showlocation=0-->\n\n          <tr>\n
          <td align="center" class="line_h24 linkf12_grey"><a href="/bzzx/cjrdw
t/2010-12-23/174.shtml">常见热点问题</a></td>\n          </tr>\n
\n\n          <tr>\n          <td align="center" class="line_h24 linkf12_grey"
><a href="/bzzx/zhmm/2010-12-23/175.shtml">找回密码</a></td>\n          </tr>\n
          \n\n          <tr>\n          <td align="center" class="line_h2
4 linkf12_grey"><a href="/bzzx/lxkf/2011-08-04/348.shtml">联系客服</a></td>\n
      </tr>\n          \n\n          <tr>\n          <td align="center"
 class="line_h24 linkf12_grey"><a href="/bzzx/hkdzl/2011-08-04/349.shtml">汇款单
招领</a></td>\n          </tr>\n          <!--/EHM_SystemNav-->\n        <
/table></td>\n      </tr>\n    </table>\n\n</div>\n</body>\n</html>\n\t\t\t\n'
>>>
```

图 3-20　交互界面返回页面内容的最后几行

将编码改为 gb2312，在访问 resp.text 时就会出现乱码，如图 3-21 所示。

```
jrdwt/2010-12-23/174.shtml">甯歌\ufffd\ufffd\ufffd\ufffd\ufffd\ufffd\ufffd归\uff
fd\ufffd棰\ufffd</a></td>\n          </tr>\n          \n\n          <tr>\n
          <td align="center" class="line_h24 linkf12_grey"><a href="/bzzx/zhmm
/2010-12-23/175.shtml">\ufffd\ufffd惧\ufffd\ufffd瀵\ufffd\ufffd\ufffd\ufffd</a><
/td>\n          </tr>\n          \n\n          <tr>\n          <td align
="center" class="line_h24 linkf12_grey"><a href="/bzzx/lxkf/2011-08-04/348.shtml
">\ufffd\ufffd\ufffd绯诲\ufffd㈢\ufffd\ufffd</a></td>\n          </tr>\n
        \n\n          <tr>\n          <td align="center" class="line_h24 linkf
12_grey"><a href="/bzzx/hkdzl/2011-08-04/349.shtml">姹\ufffd娆惧\ufffd\ufffd\uff
fd\ufffd\ufffd棰\ufffd</a></td>\n          </tr>\n          <!--/EHM_Syste
mNav-->\n        </table></td>\n      </tr>\n    </table>\n\n</div>\n</body>\n</
html>\n\t\t\t\n'
>>>
```

图 3-21　乱码现象

```
>>> resp.encoding = gb2312
>>> resp.text
```

response 具有 content 属性，可以以二进制的方式返回响应的内容，返回内容如图 3-22 所示。

```
>>> resp.content
```

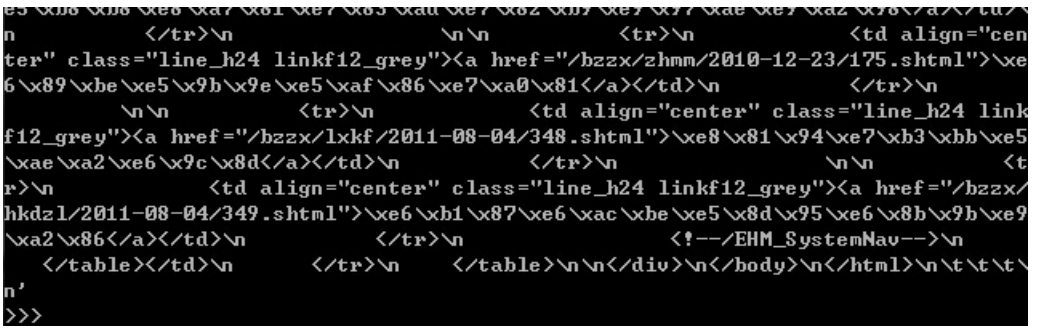

n　　</tr>\n　　\n\n　　<tr>\n　　<td align="cen
ter" class="line_h24 linkf12_grey">\xe
6\x89\xbe\xe5\x9b\x9e\xe5\xaf\x86\xe7\xa0\x81</td>\n　　</tr>\n
\n\n　　<tr>\n　　<td align="center" class="line_h24 link
f12_grey">\xe8\x81\x94\xe7\xb3\xbb\xe5
\xae\xa2\xe6\x9c\x8d</td>\n　　</tr>\n　　\n\n　　<t
r>\n　　<td align="center" class="line_h24 linkf12_grey"><a href="/bzzx/
hkdzl/2011-08-04/349.shtml">\xe6\xb1\x87\xe6\xac\xbe\xe5\x8d\x95\xe6\x8b\x9b\xe9
\xa2\x86</td>\n　　</tr>\n　　<!--/EHM_SystemNav-->\n
</table></td>\n　　</tr>\n　　</table>\n\n \n</body>\n</html>\n\t\t\t\
n'
>>>
```

图 3-22　resp.content 的返回内容

另外，response 还具有 json 方法，可以将返回的文本内容以 json 的方式进行解析。http://jsonip.com/网站接受 get 请求后，作为响应会向请求方返回一个 json 格式的文本，里面含有发起请求一方的 IP 地址，如

```
>>> resp_ip = requests.get("http://jsonip.com/")
>>> resp_ip.text
'{"ip":"43.225.237.23","about":"/about","Pro!":"http://getjsonip.com"}
>>> type(resp_ip.text)
<class 'str'>
>>> resp_ip.json()
{'about':'/about','ip':'43.225.237.23','Pro!':'http://getjsonip.com'}
>>> type(resp_ip.json())
<class 'dict'>
>>> resp_ip.text
'{"ip":"43.225.237.23","about":"/about","Pro!":"http://getjsonip.com"}
```

在该例中，resp_ip 的 text 包含返回的文本信息，因为 json 格式的信息也是以文本的方式返回的，resp_ip.text 和 type(resp_ip.text)的运行结果可以说明这个问题。resp_ip.json()使用 json 方法对返回的内容进行解析，将 json 解析的结果以字典的形式返回，但 json 方法只是解析，并不会对 resp.text 造成影响，resp.text 的内容不变。
```

以 get 进行请求时，在目标地址上会附有请求的参数，一般具有如下形式，即

```
http://www.xyz.com/method?key1=value1&key2=value2
```

对于此种类型的网址可以使用 get 方式直接发送请求，也可以将多个参数按照字典的方式组织起来发送。get 也支持这种方式，使用的方法是将“http://www.xyz.com/method”作为第一个参数，将参数字典作为第二个参数传送。前面只使用了 get 的第一个参数，即请求的地址，get 其实还有一个默认参数 params。该参数的默认值为 None，在将参数组织成字典时，将字典赋给 params 进行发送，如将上面请求中的参数组织成字典，即

```
params_dict = {
    'key1': value1,
    'key2': value2
}
```

然后使用 requests.get(http://www.xyz.com/method,params=params_dict)即可。可以使用网站“https://httpbin.org/”进行实验验证，该网站几乎覆盖了各种 http 使用场景，可以用来对各类 http 库进行测试。图 3-23 给出了该网站首页的部分截图，里面列出了提供的各种服务。

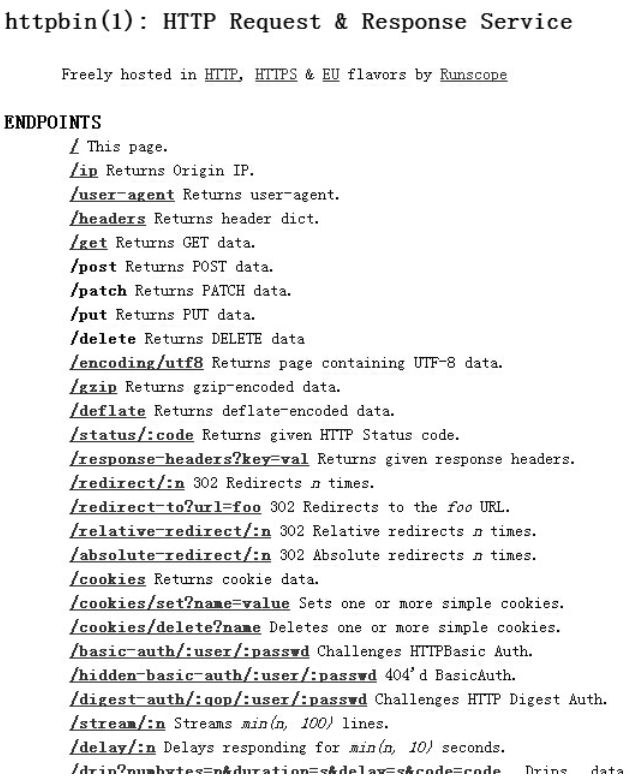


图 3-23　https://httpbin.org/提供的服务接口

下面对 requests 相关的方法进行测试，可用 response 的 json 方法将返回的 json 文本内容解析出来。

```
>>> resp1 = requests.get("http://httpbin.org/get?key1=value1&key2=value2")
>>> resp1.json()
{'origin': '43.225.237.23', 'headers': {'Host': 'httpbin.org', 'Accept': '*/*',
'User-Agent': 'python-requests/2.7.0 CPython/3.4.2 Windows/7', 'Accept-Encoding'
: 'gzip,deflate'}, 'url': 'http://httpbin.org/get?key1=value1&key2=value2', ar
gs': {'key1': 'value1', 'key2': 'value2'}}
>>> params_dict = {'key1': 'value1', 'key2': 'value2'}
>>> resp2 = requests.get("http://httpbin.org/get",params=params_dict)
>>> resp2.json()
{'origin': '43.225.237.23', 'headers': {'Host': 'httpbin.org', 'Accept': '*/*',
'User-Agent': 'python-requests/2.7.0 CPython/3.4.2 Windows/7', 'Accept-Encoding'
: 'gzip,deflate'}, 'url': 'http://httpbin.org/get?key1=value1&key2=value2', ar
gs': {'key1': 'value1', 'key2': 'value2'}}
>>> resp1.json()==resp2.json()
True
```

其中，resp1 使用的是带有参数的地址形式，resp2 使用的是将参数置于参数字典中的方式。在两种方式下，httpbin 网站都会对请求进行分析，然后返回，两种方式返回的结果相同，json 格式的 args 就是传递的参数。

有时 httpbin 会出现无法登录的情况，因为在对任何 http 进行测试时都可以利用网站，所以可以把网站搭建在本地。在 https://github.com/Runscope/httpbin 中可以下载工程源代码压缩文件，如图 3-24 所示。

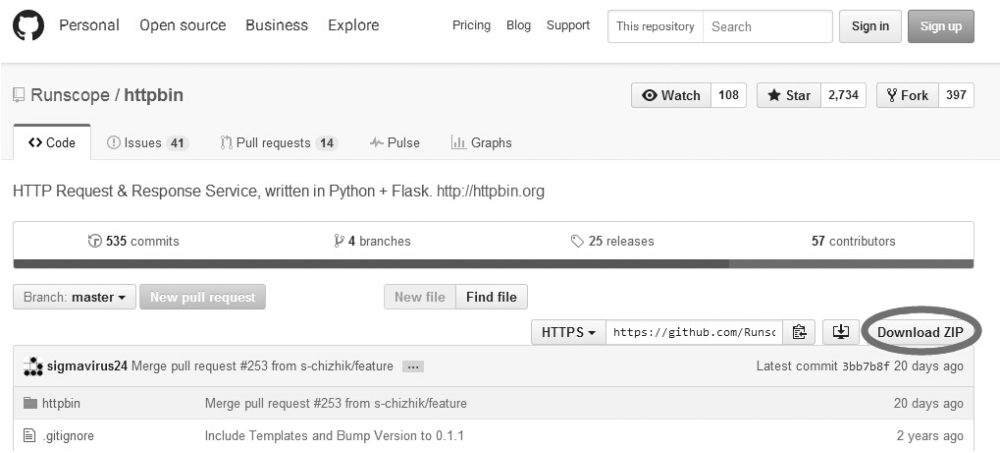


图 3-24　httpbin 下载页面

下载后解压，因为该工程源代码是使用 Python 编写的，因此在目录下直接运行 python setup. py install 进行安装即可。

也可以无需下载工程源代码，直接使用 pip install httpbin 进行安装。

安装好后，在命令行或终端运行命令 python - m httpbin. core [-- port = PORT] [-- host = HOST]即可。其中，PORT 是提供服务的端口号；HOST 是服务器地址，本地可以设置为 127. 0. 0. 1。

运行命令

```
python - m httpbin. core -- port = 9999 -- host = 127. 0. 0. 1
```

正常启动后会出现运行提示

```
 * Running on http://127. 0. 0. 1:9999/
```

之后，就可以向这个本地的服务发送请求了，如上面的 resp2 也可以发送到这个本地的服务。

```
>>> params_dict = {'key1': 'value1', 'key2': 'value2'}
>>> resp2 = requests. get("http://127. 0. 0. 1:9999/get", params = params_dict)
>>> resp2. json()
{'origin': '127. 0. 0. 1', 'headers': {'Content - Type': '', 'Content - Length': '', H
ost': '127. 0. 0. 1:9999', 'Connection': 'keep - alive', 'Accept - Encoding': 'gzip, de
flate', 'Accept': '*/*', 'User - Agent': 'python - requests/2. 7. 0 CPython/3. 4. 2 Wind
ows/7'}, 'url': 'http://127. 0. 0. 1:9999/get? key1 = value1&key2 = value2', 'args': {'k
ey1': 'value1', 'key2': 'value2'}}
```

同时注意，在启动 httpbin 服务的 shell 终端下显示该请求被收到的信息

```
 * Running on http://127. 0. 0. 1:9999/
127. 0. 0. 1 -- [10/Apr/2016 09:20:01] "GET /get? key1 = value1&key2 = value2 HTTP/1. 1"
200 -
```

2. post 方法

上面是 requests 的 get 方法介绍，requests 的另一个常用方法是 post。相比于 get 方法，post 的形式要复杂一些，因为 post 在提交时需要提供类似表单的数据信息，但 requests 对 post 和 get 方法调用类似，因此掌握起来并不困难。最常见的使用情况是将提交的数据（表单）以字典的方式组织，作为 post 的 data 参数发送。

```
>>> post_data = '{ user': 'uname', upass': 'upassword' }
>>> resp_post = requests. post( http://127. 0. 0. 1:9999/post ,data = post_data)
>>> resp_post. json( )
'{ files' :{ }', headers' :'{ Content - Type': application/x - www - form - urlencoded ',
Content - Length': 26 ', Host': 127. 0. 0. 1:9999 ', Connection': keep - alive ', Ac
cept - Encoding': gzip,deflate ', Accept': * / * ', User - Agent': python - requests
/2. 7. 0 CPython/3. 4. 2 Windows/7 }', form' :'{ upass': upassword ', user': uname
}', data': ', origin': 127. 0. 0. 1 ', url': http://127. 0. 0. 1:9999/post ', json
' :None', args' :{ } }
```

在返回 json 格式中的 form 就是提交的 post_data。可见，服务器正确解析了提交的 post 请求，也说明 post 方法提交数据成功了。这就是 post 使用的常见情况。

3. header 和 cookie

在抓取时可能对方会判断是否发送请求的一方是浏览器，如果是浏览器才进行响应，这样就需要将我们的 request 在请求时伪装成一个浏览器，通常的做法是设置 headers，其实在前面的实例中本地服务器返回的 json 串中就给出了 request 请求的 headers 信息，这个信息没有给出 headers 可能具有的全部字段，在实际抓取环境中，可以使用浏览器自带的开发控制台获得实际的 headers。

以 Firefox 浏览器为例，可以通过两种方式获得一次请求的 headers。在 3.1 节使用了 Firefox 的 Firebug 插件，为了展示解决问题途径的多样性，此处可以使用 Firefox 自带的“Web 控制台”。单击浏览器右上角的“打开菜单”图标，在弹出的窗口中单击“开发者”，如图 3-25 所示。之后在弹出的菜单中选择“Web 控制台”，如图 3-26 所示。

图 3-25　开发者工具

图 3-26　启动 Web 控制台

控制台就会出现在浏览器的下方，单击控制台的“网络”标签，如图 3-27 所示。

图 3-27　单击 Web 控制台的“网络”标签

在浏览器的地址栏中输入地址并搜索，在“网络”标签窗口就会看到发生 http 请求时，浏览器发出的 headers、参数及 cookie 信息。下面仍以电子工业出版社的网站为例来说明使用方法。在浏览器的地址栏中输入 http://www. phei. com. cn/，回车，观察控制台的窗口，如图 3-28 所示。

图 3-28　加载网页时对网络情况的监控

观察下方的窗口可以看到随着网页加载对样式表、javascript 脚本及图片等资源的请求过程，当网页加载完毕后，将左侧窗口的滚动条上移，第一条就是在浏览器内输入网址后发出的请求，即图 3-28 中的第一条请求。单击该请求，在右侧窗口将显示出该请求的详细信息。比如，在右边窗口的第一个标签“标题头”下，可以看到“请求网址”为“http://www. phei. com. cn/”，“请求头部”为

```
Host:www. phei. com. cn
User - Agent:Mozilla/5. 0 (X11;Ubuntu;Linux x86_64;rv:31. 0)
Gecko/20100101 Firefox/31. 0
Accept:text/html,application/xhtml + xml,application/xml;q = 0. 9,*/*;q = 0. 8
Accept - Language:zh - cn,zh;q = 0. 8,en - us;q = 0. 5,en;q = 0. 3
Accept - Encoding:gzip,deflate
Connection:keep - alive
```

这就是我们需要的 headers，对于其他的网站也可以使用这种方法获得请求头，获得的请求头可以组织成字典，当使用 requests 时作为 headers 参数的值，这样就可以使对方的服务器认为是浏览器发出的请求并响应。比如，对于上面请求头可以组织成字典 headers_dict，即

```
headers_dict = {
    'Host': www. phei. com. cn',
    'User - Agent': Mozilla/5. 0 (X11;Ubuntu;Linux x86_64;rv:31. 0) \
                        Gecko/20100101 Firefox/31. 0',
    'Accept': text/html,application/xhtml + xml,application/xml;\
                    q = 0. 9,*/*;q = 0. 8',
    'Accept - Language': zh - cn,zh;q = 0. 8,en - us;q = 0. 5,en;q = 0. 3',
    'Accept - Encoding': gzip,deflate',
    'Connection': keep - alive'
}
```

这样当发出下面的请求

```
requests. get("http://www. phei. com. cn/",headers = headers_dict)
```

时，在一般情况下服务器就会认为请求是由“浏览器”发出的，然后进行响应。对于本例中的网址“http://www. phei. com. cn/”，无论请求是否由浏览器发出的都会进行响应，因此可以不必考虑 headers 参数。如果服务器只对常规的浏览器发出的请求进行响应，则需要加上这个参数，否则无法得到响应。

除了请求头外，另一个在抓取时需要考虑的量就是 Cookie。与刚才获得请求头的方式相同，Cookie 值也可以使用类似的方法得到，如图 3-29 所示。单击“标题头”标签旁边的“Cookie”，即可看到请求相关的 Cookie 值。

图 3-29　Cookie 值

可以将该 Cookie 值也组织成字典的形式。其中，“JSESSIONID”为键，“728B8A3DC9-C72A07B17A315B96801249”为值。当有多个键值对时，可以依次添加到字典中。

```
cookie_dict = {
    'JSESSIONID':"728B8A3DC9C72A07B17A315B96801249"
}
```

下面的请求形式可以将请求头信息和 Cookie 值一起发出。

```
requests.get("http://www.phei.com.cn/",headers = headers_dict,cookies = cookie_dict)
```

3.3　并发和并行抓取

相比于单进程（单线程）的顺序抓取，并发或并行执行的异步抓取会极大地提高抓取效率。接下来的几节内容将主要围绕并发和并行抓取进行介绍。首先介绍并发、并行、异步和同步的概念，然后介绍 Python 中实现这些技术的主要方式，包括多线程、多进程、协程 gevent、asyncio 及 futures 等实现方式。对于每一种方式，在实施的技术细节和代码的写法上都有各自的特点。下面会举出一个抓取任务，在介绍各种方式时，会用每种方式分别去实现这个相同的任务，这样更便于比较各种方法的性能。对于各种方式，在编写程序时，代码的写法并不唯一，Python 对每种方式的实施非常灵活，下面的各节内容在实现方式上仅作为参考，感兴趣的读者可以查找手册和相关的书籍来学习并掌握更多的实现方式。

并发和并行是两个相似的概念。并发是指在一个时间段内发生若干事件的情况。并行

是指同一时刻发生若干事件的情况。可以用单核和多核 CPU 的工作方式来说明两个概念。在单核 CPU 的情况下，多任务操作系统的各任务是以并发的方式运行的，因为只有一个 CPU，所以各任务会以分时的方式在一段时间内分别占用 CPU 依次执行，如果在自己分得的时间段（又称时间片）内没有运行完成，则需等待下一次得到 CPU 的使用权时会继续执行，直至完成。在这种方式下，各任务的时间片很小，切换速度很快，所以给我们的感觉是多个任务在“同时”进行。在多核方式下，因为有两个或两个以上可以同时工作的内核，所以就有可能各核上运行的任务能够同时进行，这就是并行。

提到同步和异步的概念，一般会涉及多个任务或事件的参与。可以在并发或并行的背景下理解同步和异步。

同步指的是并发或并行发生的各任务之间不是孤立独自运行的，一个任务的进行可能需要在获得另一个任务给出的结果之后，或者只有一个任务完成或给出一个结果之后，另一个任务在获得这个结果之后才能继续运行。总之，各任务的运行会彼此相互制约，合拍前进，节奏和步调要协调好，否则就会出现错误。

异步指的是并发或并行发生的各任务之间彼此是独立运行的，不受各自的影响。这是与同步的主要区别。

当需要多个任务互相配合在并发或并行环境中合作完成时，就需要以同步的方式运行。操作系统是以信号量机制来实现同步的，各类编程语言也在进行并行或并发程序设计时提供了同步机制，Python 也是如此，如多线程设计时的锁机制。

当需要将一个大的任务分解为若干个小的子任务时，各子任务可以分别独自完成，彼此之间不必互相协作，这时就可以考虑使它们异步地并行或异步地并发执行。这在数据抓取时是一种常见的模式，将若干要抓取的链接分成几组，然后对每组分别使用子任务进行抓取，待各子任务抓取结束后，可以将结果进行汇总，完成任务。

下面介绍各种 Python 支持的并发或并行方式时，使用的抓取实例就是一个异步任务，抓取时使用 requests 库，利用不同的方式实现同一个任务有助于进行比较分析，加深对各方式的理解。每一种方式都会给出完成任务的时间，但在实际运行时，完成任务的时间和网络的情况有关。下面在介绍各方式时主要关注代码的实现方式。本章的实战部分会要求读者要在同样的网络环境下完成，并对抓取时间进行对比。

1. 多线程抓取

本节介绍多线程抓取。多线程是以并发的方式执行的，由于 Python 语言自身的设计限制，因此即使是多核的机器，Python 的多线程程序也只能运行在一个单核上以并发的方式运行。使用多线程抓取可以极大地提高抓取效率。下面以 requests 为例给出多线程抓取的实例，通过与单进程（单线程）程序比较，可以体会多线程方法在抓取效率上的提高。这个实例接下来还会在其他方式中使用。

抓取的目标为以下模式的链接，即

```
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=i&Page=2&searchKey=计算机
```

其中，在第一个 Page 处的 i 为变化的页号，本例中 i 的取值从 1 ~ 40，也就是抓取 40 个页面。

在抓取到每一个上述模式的页面后，需要进行信息的提取，将每个链接页面中所包含的指向图书详细信息页的链接提取出来，最后将所有链接保存在文本文件中。文件每行保存一本书的书名和详细信息页的相对链接，格式为“书名\t 该书的相对链接”，样式为

```
...
计算机应用基础实用教程/module/goods/wssd_content.jsp? bookid=43563
计算机实用英语(双色)/module/goods/wssd_content.jsp? bookid=43492
计算机基础实用教程/module/goods/wssd_content.jsp? bookid=43477
计算机应用基础项目教程/module/goods/wssd_content.jsp? bookid=43461
...
```

其实这与前面使用 Spynner 抓取时获得的 all_href.txt 内容是一样的。

下面的代码给出了实验所需要的函数。函数中涉及的一些原理与前面利用 Spynner 抓取时的相同，为了保持各章节内容和原理的完整性，对于一些关键的代码再一次进行说明。

```
import time
import requests
from bs4 import BeautifulSoup
import threading


def format_str(s):
    return s.replace("\n","").replace(" ","").replace("\t","")


def get_urls_in_pages(from_page_num,to_page_num):
    urls=[]
    search_word=计算机
    url_part_1=http://www.phei.com.cn/module/goods/' \
                'searchkey.jsp? Page=
    url_part_2=&Page=2&searchKey=
    for i in range(from_page_num,to_page_num+1):
```

```
        urls.append(url_part_1
                    +str(i)+
                    url_part_2+search_word)
all_href_list=[]
for url in urls:
    print(url)
    resp=requests.get(url)
    bs=BeautifulSoup(resp.text)
    a_list=bs.find_all('a')
    needed_list=[]
    for a in a_list:
        if 'href' in a.attrs:
            href_val=a['href']
            title=a.text
            if 'bookid' in href_val and 'shopcar0.jsp' \
                    not in href_val and title!='':
                if [title,href_val] not in needed_list:
                    needed_list.append([format_str(title),
                                        format_str(href_val)])
    all_href_list+=needed_list
all_href_file=open(str(from_page_num)+'_'+
                   str(to_page_num)+'_'+
                   'all_hrefs.txt','w')
for href in all_href_list:
    all_href_file.write('\t'.join(href)+'\n')
all_href_file.close()
print(from_page_num,to_page_num,len(all_href_list))
```

format_str 主要用来在提取信息后去掉多余的空白。get_urls_in_pages 函数是执行功能的主体。该函数接受两个参数 from_page_num 和 to_page_num。这两个参数确定前面所提到的链接模式中 *i* 的变化范围。在函数体中，urls 主要用来存放基于两个参数所生成的需抓取的页面链接。search_word，url_part_1 和 url_part_2 将链接模式分为 3 个部分，便于与 Page 变量的链接整合。接下来的 for 循环 for i in range(from_page_num，to_page_num + 1) 主要做的是根据参数进行拼接生成模式链接的工作，并且将每一个生成的链接加入到 urls 列表中。to_page_num + 1是由 range 取值的特点决定的，加 1 后，能够确保 to_page_num 对应的页也能被抓取到。

有了这些页面链接就可以进行下一步图书链接的抓取了。首先定义 all_href_list，这将集合未来每页中包含图书信息的列表，从后面的代码可以看到该列表其实是“列表的列表”，因为 all_href_list 中的每个元素均为列表，形式为[书名 ', 详细页相对链接]，所以在每页可以提取出图书信息的情况下，all_href_list 将具有以下形式

```
all_href_list = [
    [ 书名 1 ', 相对链接 1 ],
    [ 书名 2 ', 相对链接 2 ],
    [ 书名 3 ', 相对链接 3 ],
    [ 书名 4 ', 相对链接 4 ],
    ......
]
```

接下来的代码就是对每一页进行抓取和信息的抽取了。这部分代码在 for url in urls 循环体中，首先在 for 循环中打出本次遍历的 url 字符串，这样做更加便于调试。在 requests 利用 get 或 post 方式抓取时，在出现错误的情况下，可以方便地看出是不是 url 参数的格式出现了错误，因为 url 是通过几个部分的组合得到的，在字符串的连接时可能出现错误。另外，在后面进行多线程和单线程对比时，可以通过 url 的打印情况加深对多线程执行异步性的认识。在对 url 指示页面进行抓取时，无需提供表单或数据信息，因此直接使用 requests 的 get 方法即可。为了方便地提取页面的图书名称和相对链接，使用 BeautifulSoup 将 get 请求返回的 HTML 文本进行分析，转为 BeautifulSoup 能够处理的结构，这里命名为 bs。

使用 requests 的 get 方法得到每次循环所遍历到的 url 链接对应的页面，每页中包含的书名和相对链接保存在 needed_list 中，needed_list 和 all_href_list 的格式类似。在对页面中的链接进行提取时使用 BeautifulSoup 中的相关方法。bs. find_all('a')抽取了页面中的所有链接元素，for a in a_list 对每一个列表中的元素进行遍历分析，a['href']和 a. txt 分别可以获得一个链接元素的链接信息和文本信息。这里直接使用 a['href']，对于其他网站，可能设计时并不是每个链接元素都有 'href' 属性，这时需要判断 'href' 是否在元素 a 的属性中，if 'href' in a. attrs 就起到这个作用。这里用到了 BeautifulSoup 中元素的 attrs 属性。接下来的 if 判断

```
if 'bookid' in href_val and 'shopcar0. jsp' \
                    not in href_val and title != '':
```

主要是将希望得到元素 a 的信息提取出来。我们希望得到链接元素的 href 属性中含有 'bookid' 且不含有 'shopcar0. jsp'，并且同时书名 title 不能为空串，这是根据具体分析得到的筛选条件。因为有时在页面中一个对象不但有文字链接，同时配有的图片也有相应的链接，

而且它们是相同的，所以在一个页面中分析出一个有效的链接元素 a 后，在加入该页的 needed_list 时需要判断是否元素 a 的信息已经存在，若不存在，才加入

```
if [title,href_val] not in needed_list:
```

起到的就是这个作用，每抽取完一个页面的链接后，就可以把它加入到 all_href_list 中，即

```
all_href_list += needed_list
```

注意使用的是 += 运算符。

最后收集到 from_page_num 至 to_page_num 的所有链接后，就可以写入文件了，后面的代码执行这个功能，并输出此次抓取的相关数据值。

这就是完成抓取任务的主要代码。下面通过对比实验来比较多线程和单线程之间抓取效率的差异。抓取的任务是抓取连续的 40 页，并将其中所包含的所有链接提取出来。在多线程实现方案中，这 40 页的抓取和分析任务由 4 个线程来完成，其中每个线程处理 10 个页面的任务。在单进程（单线程）的情况下，这 40 页将由一个线程单独抓取完成。

首先设计多线程的抓取方案，将其置于函数 multiple_threads_test 中，代码为

```
def multiple_threads_test():
    t1 = time.time()
    page_ranges_lst = [
        (1,10),
        (11,20),
        (21,30),
        (31,40),
    ]

    th_lst = []
    for page_range in page_ranges_lst:
        th = threading.Thread(target = get_urls_in_pages,
                              args = (page_range[0],page_range[1]))
        th_lst.append(th)

    for th in th_lst:
        th.start()

    for th in th_lst:
        th.join()
```

```
    t2 = time. time( )
    print(' 使用时间 1': ,t2 - t1)
    return t2 - t1
```

t_1 记录了实验的开始时间。page_ranges_lst 是一个列表，列表中的元素是元组，每个元组给出了一次抓取时的页号范围，对应于前面定义的 get_urls_in_pages 函数参数，其实这个函数也是下面创建线程时使用的线程目标函数。th_lst 是线程列表，主要用于存放创建完的线程。接下来的循环基于 page_ranges_lst 的 4 个元素创建 4 个线程，即

```
th = threading. Thread( target = get_urls_in_pages,
                                    args = ( page_range[ 0] , page_range[ 1] ) )
```

使用 threading 模块的 Thread 类来初始化一个线程实例，target 是线程的活动实体，未来线程将要执行的主要工作由该参数指定的函数完成。注意，参数值是 get_urls_in_pages 函数的函数名，不是 get_urls_in_pages( )，带括号的形式表示对函数的调用。如果写成 target = get_urls_in_pages( )，表示调用 get_urls_in_pages 函数，并将执行结果赋给 target。这并不是我们的意图，同时此处这样调用也是错误的，因为 get_urls_in_pages 函数在使用时要接受两个参数。总之应该注意，target 的值应该是函数名，而不是函数的调用结果。在 target 的值正确指定后，接下来就是指定第二个参数 args 了。args 的值应为元组类型，如此处(page_range[0] 、page_range[1])恰为一个元组，需要注意的是，target 所传入的函数名在调用时接受 1 个参数 arg1，args 应传入(arg1,)的形式。因为若写成 args = (arg1)，则 Python 会认为 args = arg1 是一个整数，这与 Python 对整数的表示方法有关，如图 3-30 所示。

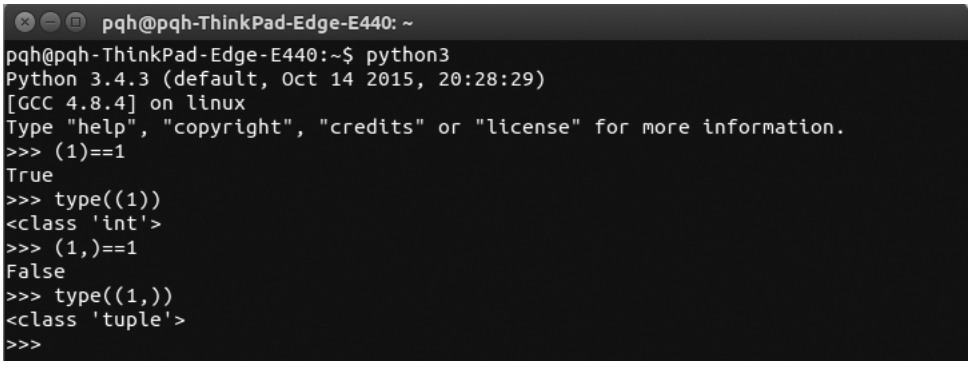


图 3-30　（1）和（1,）的区别

接下来看下一句

```
    th_lst. append( th)
```

将生成的线程都放入 th_lst 列表中，此时 th 只是线程初始化后的实例，还未开始执行。在接下来的循环中，对 th_lst 列表中的线程分别执行 th. start( )，此后这些线程就会在系统中接受系统的调度，异步并发地执行了。为了使这些异步并发执行的线程都执行完毕后再退出外层函数 multiple_threads_test，这里使用了线程的 join 方法，等待各线程执行完毕。最后，记录下所有线程执行完成的时间 t_2，$t_2 - t_1$，可以得到利用线程方式完成任务的全部时间。下面进行测试

```
if __name__ == '__main__':

    mt = multiple_threads_test( )
    print('mt',mt)
```

为了表明多线程的并发执行效果，进行了两次运行实验，并将两次执行时输出的开始部分列出进行对比。

第一次实验的部分输出为

```
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=1&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=11&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=21&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=31&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=32&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=22&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=33&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=23&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=2&Page=2&searchKey=计算机
......
```

第二次实验的部分输出为

```
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=1&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=11&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=21&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=31&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=2&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=12&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=22&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=32&Page=2&searchKey=计算机
```

```
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=13&Page=2&searchKey=计算机
.....
```

由第一个 Page 参数的取值可以看出，多线程执行方法是以异步方式进行的，多个线程之间在系统调度下异步并发执行，在每一个线程内部按照所分配的抓取任务依次按序执行。如果多进行几次运行，就会发现开始部分的输出列表可能还会有所变化，但线程间异步并发和线程内部按顺序执行的性质保持不变。

第一次实验完成时总用时为 61.29 秒，第二次总用时为 63.04 秒。时间主要是由网络情况决定的，一方面取决于服务器的响应时间，另一方面取决于抓取时机器所在局域网的网络情况。下面进行单线程（单进程）实验。在相同的网络情况下，可以预计单线程所用的时间将是 4 个多线程执行时间的 4 倍左右。单线程的实验代码为

```
def single_test():
    t1 = time.time()
    get_urls_in_pages(1,40)
    t2 = time.time()
    print('使用时间 2':,t2 - t1)
    return t2 - t1
```

在单线程实验中，直接调用 get_urls_in_pages，并传递参数 1、40，在代码中做如下改动，将刚才的多线程实验函数注释掉，改为

```
if __name__ == '__main__':

    # mt = multiple_threads_test()
    # print('mt',mt)

    st = single_test()
    print('st',st)
```

运行后，输出的开始部分打印出的字符串为

```
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=1&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=2&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=3&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=4&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp? Page=5&Page=2&searchKey=计算机
```

```
http://www.phei.com.cn/module/goods/searchkey.jsp?Page=6&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp?Page=7&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp?Page=8&Page=2&searchKey=计算机
http://www.phei.com.cn/module/goods/searchkey.jsp?Page=9&Page=2&searchKey=计算机
......
```

在此种方式下，程序会顺序依次抓取每一个链接，直至任务完成。最后的用时约为212.10 秒，为多线程执行方式的 3 倍多。可见，多线程执行方式的确能够提高抓取的效率。

2. 多进程抓取

Python 多线程程序只能运行在单核上，在上节的实例中，各线程是以并发的方式异步运行的。本节介绍多进程方式。多进程也是提高抓取效率的有效手段。多进程方式依赖于所在机器的处理器个数。在多核机器上进行多进程编程时，各核上运行的进程之间是并行执行的。可以利用进程池，使每一个内核上运行一个进程，当池中的进程数量大于内核总数时，待运行的进程会等待，直至其他进程运行完毕让出内核。当系统内只有一个单核 CPU 时，多进程并行不会发生，此时各进程会依次占用 CPU 运行至完成。下面的代码给出了在 8 核处理器上运行多进程的程序，可以使用下面的语句获得 CPU 可用的核数，如图 3-31 所示。

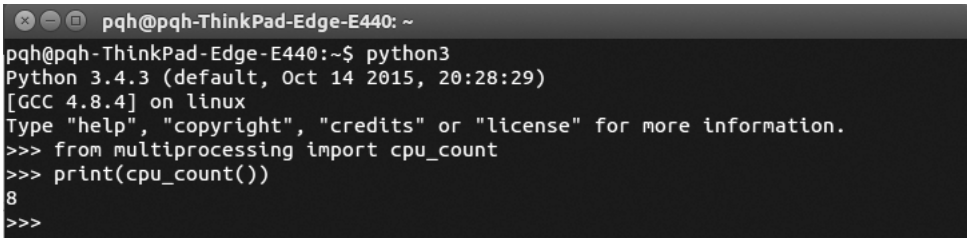


图 3-31　CPU 核数

为了与前面多线程进行比较，这里使用其中的 4 个核，在使用进程池时，4 作为参数传入。在前面多线程程序中加入

```
import multiprocessing
```

然后定义测试函数 multiple_processes_test 即可，即

```
def multiple_processes_test():
    t1 = time.time()
    page_ranges_lst = [
        (1,10),
```

```
        (11,20),
        (21,30),
        (31,40),
    ]

    pool = multiprocessing.Pool(processes=4)
    for page_range in page_ranges_lst:
        pool.apply_async(get_urls_in_pages,
                         (page_range[0],page_range[1]))

    pool.close()
    pool.join()
    t2 = time.time()
    print('使用时间:',t2-t1)
    return t2-t1
```

其中，pool 被定义为可同时并行 4 个进程的进程池，apply_async 方法可以使进入进程池的进程以异步的方式并行运行，即

```
if __name__=='__main__':

    #mt = multiple_threads_test()
    #print('mt',mt)

    #st = single_test()
    #print('st',st)

    pt = multiple_processes_test()
    print('pt',pt)
```

运行代码，因为 multiple_processes_test 与前面的多线程 multiple_threads_test 测试函数和单线程 single_test 函数写在了一个文件中，且当前这两个函数已经被注释掉，因此直接运行就可以了，在运行结果中可以看到

```
pt 12.9848313331604
```

注意，该时间值和网络情况有关，如果希望与前面的多线程进行比较，则可以取消多线程方式的注释，使它们依次运行，这样可以认为它们在相同的网络环境下运行，才具有

可比性。

3. 利用协程抓取

除了上面介绍的多线程和多进程外，Python 还支持协程（coroutine）编程。一般在 Python 讨论协程时，都会与生成器（generator）联系在一起。生成器是一个函数，主要特点是生成器在返回值时，不是使用 return，而是使用 yield 关键字，在定义函数时，如果函数体中包含 yield 关键字，则该函数就被认为是一个生成器。

本节不过多地对生成器的写法进行介绍，而是直接引入 gevent 库，使用 gevent 异步库可以方便地实现基于协程的并发设计。可以利用 gevent 的“monkey patch”能力，将标准的 I/O 函数转为异步执行的函数，而代码基本不用变化。在 gevent 中使用 greenlet 对象实现并发，greenlet 就是协程，可以认为是一种轻量线程。

可以使用下面的命令安装 gevent，即

```
sudo pip3 install gevent
```

安装完成后，就可以使用了。下面是使用协程进行抓取的代码，保存在 gevent_test. py 中，即

```
import gevent
from gevent import monkey
monkey. patch_all( )
import requests
from bs4 import BeautifulSoup
import time

def format_str(s):
    return s. replace("\n","" ). replace(" ","" ). replace("\t","" )

def get_urls_in_pages(from_page_num,to_page_num):
    ......
该段代码与前面多线程和多进程抓取时使用的一致。
......

def gevent_test( ):

    t1 = time. time( )
```

```
    page_ranges_lst = [
        (1,10),
        (11,20),
        (21,30),
        (31,40),
    ]

    jobs = []
    for page_range in page_ranges_lst:
        jobs.append(gevent.spawn(get_urls_in_pages,
                                 page_range[0],page_range[1]))

    gevent.joinall(jobs)

    t2 = time.time()
    print('使用时间',t2 - t1)
    return t2 - t1

if __name__ == '__main__':
    gevent_test()
```

这段代码要注意

```
from gevent import monkey
monkey.patch_all()
```

如果没有这两句，则会变为依次顺序抓取，失去了并发的能力。

gevent.spawn 可以生成 greenlet，gevent.joinall(jobs)，会阻塞程序的执行，直至所有的协程运行完毕。

执行这段代码，可以看到运行时间约为 10s。

4. 利用 asyncio 库抓取

从 Python3.4 开始，Python 自身提供了 asyncio 库用于实现异步 I/O 程序设计，利用 asyncio 可以使用协程进行并发编程，在使用 asyncio 时，除了涉及前面介绍的协程，还有事件循环（event loop）。事件循环的作用是调度各协程运行。asyncio 提供了对套接字（socket）API 的全面支持，但在使用时涉及的细节问题比较多，可以引入另一个库 .aiohttp。该库基于 asyncio 提供了 http 服务器和客户端编程的支持，可以方便地发出 http 请求。这一

点与前面编程时引入 requests 的原因是类似的，都提供了更易于理解和操作的方法调用来完成相应的工作。

安装 aiohttp 的命令为

```
sudo pip3 install aiohttp
```

下面给出了使用 asyncio 和 aiohttp 实现异步并发抓取的代码，并保存在 asyncio_test.py 文件中。这段代码给出了使用 asyncio 的一种方法。下面主要围绕 asyncio 在编程使用时的一些设计特征对这段代码进行说明。注意，get_urls_in_pages 与前面介绍的内容功能是相同的，但因为程序使用了 asyncio 方式进行设计，所以在一些位置代码会有所调整。

```
import asyncio
import aiohttp
import time
from bs4 import BeautifulSoup

sem = asyncio.Semaphore(10)

def decoder(content):
    return content.decode('utf-8')

def format_str(s):
    return s.replace("\n","").replace(" ","").replace("\t","")

@asyncio.coroutine
def get_urls_in_pages(from_page_num,to_page_num):
    urls = []
    search_word = 计算机
    url_part_1 = http://www.phei.com.cn/module/goods/\
                 'searchkey.jsp?Page=
    url_part_2 = &Page=2&searchKey=
    for i in range(from_page_num,to_page_num+1):
        urls.append(url_part_1
                    +str(i)+
                        url_part_2+search_word)
    all_href_list = []
    for url in urls:
```

```
            print(url)
            # resp = requests.get(url)
            with (yield from sem):
                response = yield from aiohttp.request('GET',url)
            body = yield from response.read_and_close()
            html_str = decoder(body)
            bs = BeautifulSoup(html_str)
            a_list = bs.find_all('a')
            needed_list = []
            for a in a_list:
                if 'href' in a.attrs:
                    href_val = a['href']
                    title = a.text
                    if 'bookid' in href_val and 'shopcar0.jsp' \
                            not in href_val and title != '':
                        if [title,href_val] not in needed_list:
                            needed_list.append([format_str(title),
                                                format_str(href_val)])
            all_href_list += needed_list
        all_href_file = open(str(from_page_num) + '_' +
                             str(to_page_num) + '_' +
                             'all_hrefs.txt','w')
        for href in all_href_list:
            all_href_file.write('\t'.join(href) + '\n')
        all_href_file.close()
        print(from_page_num,to_page_num,len(all_href_list))


def asyncio_test():
    t1 = time.time()
    page_ranges_lst = [
        (1,10),
        (11,20),
        (21,30),
        (31,40),
    ]
```

```
    loop = asyncio. get_event_loop( )

    f = asyncio. wait( [ get_urls_in_pages( page_range[ 0 ] , page_range[ 1 ] )
                        for page_range in page_ranges_lst ] )

    loop. run_until_complete( f )

    t2 = time. time( )
    print( '使用时间': , t2 - t1 )
    return t2 - t1


if __name__ = = '__main__' :
    at = asyncio_test( )
    print( 'at': , at )
```

首先引入了

```
import asyncio
import aiohttp
```

这两个库。其中，asyncio 主要提供了异步并发编程的框架，而 aiohttp 用于在抓取时发出 http 请求。下一句

```
sem = asyncio. Semaphore( 4 )
```

的作用是通过调用 asyncio 的 Semaphore 方法设置一个信号量 sem，通过该信号量可以控制未来在 asyncio 异步框架中同时可发出 http 请求的协程数目。本例中传入 Semaphore 的参数是 4，也就是说，在抓取时最多并发发出 4 个请求。

接下来定义了两个函数，其中

```
def decoder( content) :
    return content. decode( 'utf - 8' )
```

的作用是解码，将字节码使用 utf8 进行解码。另一个 format_str 的主要作用是去掉多余的空白字符。

下一句

```
@ asyncio. coroutine
```

是 Python 的修饰器语法形式。在本例中，该句位于 get_urls_in_pages 之前，将 get_urls_in_pages 修饰成一个协程。在接下来的代码中，比较有特点的就是 get_urls_in_pages 中用于抓取的代码了，如

```
with (yield from sem):
        response = yield from aiohttp. request('GET',url)
body = yield from response. read_and_close()
```

其中，with 语法形式在前面的文件操作时曾经引入过，在这里的基本作用并没有改变，都是为使用前需要设置、使用后需要清理的使用场景提供一种简洁的使用模式。这三句都使用了

```
yield from
```

语句，是 Python3. 3 之后新增的语法形式，可以从一个协程中调用另一个协程。通过前面的介绍可知本例中最多可发出 4 个网页请求，with(yield from sem)就起到了这个数量控制作用，与操作系统原理中临界区和信号量概念类似。如果与之类比的话，则

```
response = yield from aiohttp. request('GET',url)
```

可理解为临界区，而 with(yield from sem)就相当于进入临界区之前进入区内的代码，只有满足进入条件的协程才会通过代码 with(yield from sem)，从而进入临界区发出请求。如果条件不满足，就会在 with(yield from sem)处发生阻塞，不会进入临界区，直至条件满足时被唤醒，再进入临界区执行。这里的条件是 asyncio 库初始化和维护的 sem 对象决定的，在初始化时已经传入了参数 4，那么未来在运行时，最多有 4 个协程可以进入临界区发出请求。在该句中

```
yield from aiohttp. request('GET',url)
```

的主要功能是发出 http 请求，并在此处等待请求的结果，当结果返回后将其赋值给 response，发出请求直至完成会有比较长的网络 I/O 延时，但在本例中使用了 asyncio 和 aiohttp 库，同时利用 Python 的 yield from 语法，可以使运行时的协程发出请求后，在等待响应的过程中，不会阻塞程序的执行。asyncio 的事件循环调度机制会继续运行其他协程，提高整体运行效率。

aiohttp 库的请求方法返回的响应对象也支持 yield from 的调用机制，下一句

```
body = yield from response. read_and_close( )
```

表示将从对象中读取内容，并将内容赋给 body 变量。注意，返回的内容是二进制形式的，需要进行解码。get_urls_in_pages 后面的代码就与前面的几节定义一样了，使用 html_str = decoder(body) 解码后，对解码得到的文本信息进行抽取和保存。

接下来的 asyncio_test 函数主要实现利用 asyncio 事件循环机制来调度定义的协程并发运行。

```
loop = asyncio. get_event_loop( )
cs = asyncio. wait( [ get_urls_in_pages( page_range[0] ,page_range[1] )
                                for page_range in page_ranges_lst] )
loop. run_until_complete( cs)
```

其中，第一句 asyncio. get_event_loop( ) 可以获得事件循环对象 loop，loop 的使用是在第三句，调用了 loop 的 run_until_complete 方法。从名字可以看出，该方法表示事件循环机制会运行直至完成。循环机制运行调度的是诸协程，由参数 cs 传入。cs 是由 asyncio. wait 生成的。asyncio. wait 接受一个协程的列表作为参数。上面的三句代码完成的主要功能就是启动事件循环机制，生成待运行的诸协程，然后调度运行。

执行这段代码耗时 10 秒左右。

5. 利用 futures 库抓取

Python 3 中引入了 concurrent. futures 库。该模块提供了高级接口，用于实现异步任务的执行。异步任务可以使用 ThreadPoolExecutor 以多线程的方式执行，或者使用 ProcessPoolExecutor 以多进程的方式执行。这两者都实现了抽象类 Executor 定义的接口，在使用方式上风格类似。下面给出利用 concurrent. futures 实现的异步抓取，分别给出 ThreadPoolExecutor 和 ProcessPoolExecutor 的使用实例。实例与前面的一样，因此在下面的代码中省略了较长 get_urls_in_pages(from_page_num,to_page_num) 函数的内容。

在代码开头引入需要的库

```
from concurrent import futures
```

就可以了。下面的函数 futures_threads 和 futures_processes 分别是多线程和多进程的实现，即

```
def futures_threads( ) :
    t1 = time. time( )
    page_ranges_lst = [
        (1,10),
```

```
            (11,20),
            (21,30),
            (31,40),
        ]
        with futures.ThreadPoolExecutor(max_workers=4) as executor:
            future_to_url = {executor.submit(get_urls_in_pages,
                                             page_range[0],
                                             page_range[1]):ind
                             for ind,page_range in
                             enumerate(page_ranges_lst)}

            for future in futures.as_completed(future_to_url):
                range_ind = future_to_url[future]
                if future.exception() is not None:
                    print('%d generated an exception:%s' %
                          (range_ind,future.exception()))
                else:
                    print('%d get urls is %d' %
                          (range_ind,future.result()))

        t2 = time.time()
        print('threads':,t2 - t1)

    def futures_processes():
        t1 = time.time()
        page_ranges_lst = [
            (1,10),
            (11,20),
            (21,30),
            (31,40),
        ]
        with futures.ProcessPoolExecutor(max_workers=4) as executor:
            future_to_url = {executor.submit(get_urls_in_pages,
                                             page_range[0],
                                             page_range[1]):ind
                             for ind,page_range in
```

```
                enumerate(page_ranges_lst)}

            for future in futures.as_completed(future_to_url):
                range_ind = future_to_url[future]
                if future.exception() is not None:
                    print('%d generated an exception:%s' %
                          (range_ind,future.exception()))
                else:
                    print('%d get urls is %d' %
                          (range_ind,future.result()))

        t2 = time.time()

        print('processes':,t2 - t1)


if __name__ == '__main__':

    futures_threads()
    futures_processes()
```

代码中，futures_threads()函数用来测试 ThreadPoolExecutor，futures_processes()用来测试 ProcessPoolExecutor。使用 ThreadPoolExecutor 时，传入参数 max_workers = 4，那么未来在线程池中，异步执行的线程最多将有 4 个，ThreadPoolExecutor 在使用时需要传入 max_workers 参数的值。使用 ProcessPoolExecutor 时，传入的参数也为 max_workers = 4，那么未来在进程池中，异步执行的进程个数最多为 4 个。ProcessPoolExecutor 在使用时也可以不必提供参数，默认 max_workers 参数的个数为机器所在处理器的个数。

实战

将本节介绍的各种并行或并发模式部署运行，对所有的运行模式适当地修改程序，分别抓取 20 页、40 页、60 页、80 页，对比在不同模式、相同页数下的不同抓取时间，并做出对比曲线。曲线的做法和对比的形式可参考 5.1 节的 2 “对 Selenium 的 profile 配置”及图 5-10。

4 第4章 分布式数据抓取

分布式数据抓取的思想是在分布式网络中，将抓取任务分派到各个抓取节点，由各个抓取节点进行抓取。抓取之后或将抓取结果，如 HTML 文件、图片、文档等保存在本地，或将抓取结果返回，也可以对抓取结果的信息抽取和处理之后再返回。

在完成抓取任务时，抓取效率是一个关键。对于一个一般的抓取任务，从网络环境的角度考虑，如果抓取机器的网络带宽较高，如 1MB/s、512KB/s，或者说常规的 ADSL 提供的带宽，就可以满足要求了。从抓取机器的角度来讲，一般的台式机即可。因为抓取的主要瓶颈在网络环境，如上面所说的带宽问题，或者目标网站的响应延时等。抓取机器的主要工作是合理地调度抓取任务，包括设计多线程、多进程的抓取策略，协调抓取的各个环节，如抓取数据和处理数据环节的衔接等。这些工作占用的内存不大，通常 4G 以上即可满足一般抓取的需求。抓取不属于计算密集型任务，对 CPU 也没有太高要求，但需要为保存抓取的数据、图片音频或视频等，所以需要较大的磁盘空间。

通过上面的分析可以看出，一台机器在网络环境畅通的情况下即可完成常规的任务。那么，为什么需要分布式抓取方法呢?

（1）提高抓取效率

在进行分布式抓取时，各抓取节点的 IP 不同，具有各自独立的公网 IP 地址，原来由一台机器完成的任务被分散到多台机器上，就可减小抓取任务被网站封锁的可能性。同时，因为在各个抓取机器上的任务是并行执行的，因此总抓取任务完成的时间就会缩短，抓取效率也会因此提高。需要说明的是，这里强调的是每台抓取机器最好具有独立的公网 IP。如果搭建在局域网环境下，那么多台机器的内网 IP 不同但公网 IP 相同，这样尽管任务的执行方式不变，但目标网站只会看到一个 IP。如果网站设置了封锁机制，那么这种抓取方式下被封的可能性和在一台机器上使用多线程或多进程等方式被封锁的可能性是一样的。

（2）数据分析的需要

数据抓取是为数据分析服务的，可以将各节点抓取的数据直接返回到分派该抓取任务的服务器，也可以抽取出信息之后再返回，当然也可以保存在本地。保存在抓取节点本地的好处是，可以在未来有需要时直接在抓取端进行数据的提取和分析，然后将结果返回，

类似于 MapReduce 的工作方式，因此也可以为分布式数据分析框架提供数据支持。

（3）周期性任务的需要

有时需要抓取的数据是以一定周期更新的，抓取需要按时将这些信息抓取下来：一方面是为数据分析提供及时的数据；另一方面是有些信息在一段时间后可能不易获取。这样就需要设计周期性的抓取任务及时地抓取。这种无需人工监控的周期性任务就可以部署在分布式抓取节点上，避免了本地机器在使用过程中开机、关机或执行其他任务对抓取造成的影响。

下面介绍两种分布式数据抓取任务的实施方法，分别是利用 RPC 和 Celery。RPC 和 Celery 既可以在 Windows 平台部署使用，也可以在 Linux 平台上部署使用。本章兼顾这两种情况，在实施时，将 RPC 服务部署在运行 Windows 系统的云端，将 Celery 服务部署在 Linux 云端。

RPC 服务的实例使用了 4 台具有独立 IP 的阿里云端作为 RPC 服务器进行抓取，本地计算机作为分派任务的服务器。4 台阿里云端配置相同，操作系统为 Windows 2008 server 服务器，都具有 1G 的内存，单核 2.3G 处理器，10MB/s 带宽和 40G 的硬盘。硬盘只有一个 C 分区，且已经安装了操作系统，因此可用空间为 30 多个 G。

Celery 服务的实例使用了 4 台具有独立 IP 的阿里云端，4 个云端的物理配置与 RPC 服务部署的机器一样，但运行的是 Ubuntu 14.04 trusty 操作系统。

4.1 RPC 的使用

远程过程调用（Remote Procedure Call，RPC）是在本地调用远程方法的技术。这种技术用途广泛，可以应用于各种场景解决问题。本节将使用 RPC 技术实施分布式数据抓取。Python 自带了 PRC 实施模块 xmlrpc。

RPC 的思想正好可以用来实现分布式抓取框架。基本思想如下：在云端机或本地局域网内的机器运行 RPC 服务器，服务器提供根据需要编写好的函数供调用，可以编写各种函数，如抓取函数。需要抓取时，只要编写好程序，则调用这些服务器提供的函数即可，就好像在本地调用一样。首先给出 RPC 服务端代码，文件名为 rpc_server.py，将这些代码部署到前面提到的 4 台具有独立 IP 阿里云端的抓取节点上。如果读者没有云端，也可以在本地局域网内实验。代码为

```
from xmlrpc.server import SimpleXMLRPCServer
import socketserver
import requests
```

```
import sys

class Crawler:
    def get(self,user,url,params = None,headers = None):
        try:
            if user == username:
                r = requests.get(url,params = params,headers = headers)
                return r.text
            else:
                return''
        except Exception as e:
            return e

    def post(self,user,url,data = None,headers = None):
        try:
            if user == username:
                r = requests.post(url,data = data,headers = headers)
                return r.text
            else:
                return''
        except Exception as e:
            return e

if__name__ == __main__:
    ip = sys.argv[1]
    port = sys.argv[2]

    class RPCThreading(socketserver.ThreadingMixIn,SimpleXMLRPCServer):
        pass
    crawler_object = Crawler()
    server = RPCThreading((ip,int(port)))
    server.register_instance(crawler_object)

    print("Listening")
    server.serve_forever()
```

可以从 if __name__ == __main__ 下的代码开始解读。其中，

```
ip = sys. argv[1]
port = sys. argv[2]
```

从命令行得到 rpc_server. py 运行时机器所在的 IP 和监听端口。语句

```
class RPCThreading(socketserver. ThreadingMixIn,SimpleXMLRPCServer):
    pass
```

定义了 RPCThreading 类，通过这个类，可以将 RPC 服务器工作时的单任务模式转换为多线程模式。这样在服务器被频繁调动时可以提高并发度，使用这个类来初始化 RPC 服务器对象。下一句

```
crawler_object = Crawler()
```

初始化 Crawler 生成 crawler_object 对象。在类 Crawler 中定义了两个方法，get 和 post。这两个方法从定义的形式上与第 1 章 1.4 节第 4 部分介绍的 Python 面向对象编程中的定义形式是相同的，从内容看，两个方法是对 requests 中 get 和 post 方法的封装，而且加入了简单的验证机制，即

```
user == username :
```

' username' 是 RPC 服务端定义的用户名。RPC 服务请求要携带用户名参数，只有用户名通过验证，才会进一步提供 requests 的相应服务，并将抓取内容返回。如果不能通过验证，则返回空串。下面两句

```
server = RPCThreading((ip,int(port)))
server. register_instance(crawler_object)
```

第一句的主要作用是初始化 RPCThreading 类，生成可以运行的 server 服务器，在初始化时传入了 IP 和 port，未来这个服务器将一直运行监听 port 端口。第二句的作用是将 crawler_object 对象注册到 server 上，未来有 RPC 请求到达 port 端口时，server 会根据请求的内容调用 crawler_object 中相应的方法执行并将结果返回。最后一句为

```
server. serve_forever()
```

执行完此句后，RPC 服务器就启动了。

抓取程序运行机器的代码命名为 rpc_crawler. py。

```
from xmlrpc. client import ServerProxy
```

```
from bs4 import BeautifulSoup
import threading
import time


def get_server_ips(sfile):
    server_ips = []
    with open(sfile,'r',encoding='utf8') as f:
        for ind,line in enumerate(f.readlines()):
            ip = line.strip()
            server_ips.append([ind,ip])
    return server_ips


def format_str(s):
    return s.replace("\n","").replace(" ","").replace("\t","")


def get_urls_in_pages(from_page_num,to_page_num,remote_ip):
    server = ServerProxy(remote_ip[1])
    urls = []
    search_word = '计算机'
    url_part_1 = 'http://www.phei.com.cn/module/goods/' \
                 'searchkey.jsp?Page='
    url_part_2 = '&Page=2&searchKey='
    for i in range(from_page_num,to_page_num+1):
        urls.append(url_part_1
                    +str(i)+
                    url_part_2+search_word)
    all_href_list = []
    for url in urls:
        print(remote_ip[1],url)
        #resp = requests.get(url)
        rstr = server.get('username',url,{},{})
        bs = BeautifulSoup(rstr)
        a_list = bs.find_all('a')
        needed_list = []
        for a in a_list:
            if 'href' in a.attrs:
```

```
                    href_val = a['href']
                    title = a.text
                    if 'bookid' in href_val and 'shopcar0.jsp' \
                            not in href_val and title!='':
                        if [title,href_val] not in needed_list:
                            needed_list.append([format_str(title),
                                                format_str(href_val)])
        all_href_list += needed_list
    all_href_file = open(str(from_page_num) + '_' +
                         str(to_page_num) + '_' +
                        'all_hrefs.txt','w')
    for href in all_href_list:
        all_href_file.write('\t'.join(href) + '\n')
    all_href_file.close()
    print(from_page_num,to_page_num,len(all_href_list))

def multiple_threads_test():
    server_ips = get_server_ips('remotes')
    server_cnt = len(server_ips)
    t1 = time.time()
    page_ranges_lst = [
        (1,10),
        (11,20),
        (21,30),
        (31,40),
    ]

    th_lst = []
    for ind,page_range in enumerate(page_ranges_lst):
        th = threading.Thread(target = get_urls_in_pages,
                              args = (page_range[0],page_range[1],
                                      server_ips[ind % server_cnt]))
        th_lst.append(th)

    for th in th_lst:
        th.start()
```

```
        for th in th_lst:
            th.join()


        t2 = time.time()
        print('使用时间 1:',t2 - t1)
        return t2 - t1


if __name__ == '__main__':

        mt = multiple_threads_test()
        print("mt ",mt)
```

抓取的代码是根据 3.3 节多线程抓取中的代码修改过来的，最大的改变就是在前面的多线程抓取。每个线程抓取都是在本机调用 requests 的 get 方法。本例每个线程抓取时将调用某个远程 RPC 服务器上提供的抓取方法。首先，

```
from xmlrpc.client import ServerProxy
```

引入了 ServerProxy，未来可以连接远程 RPC 服务器，并请求服务。

get_server_ips 函数接收一个包含服务器 IP 地址列表的文件，然后读取并返回包含由[IP 地址，索引值]组成的列表。假设服务器 IP 存放的文件为 remotes，内容为

```
xxx.xxx.xxx.111
xxx.xxx.xxx.112
xxx.xxx.xxx.113
xxx.xxx.xxx.114
```

那么 get_server_ips('remotes')返回的结果为

```
[
    [0,'xxx.xxx.xxx.111'],
    [1,'xxx.xxx.xxx.112'],
    [2,'xxx.xxx.xxx.113'],
    [3,'xxx.xxx.xxx.114']
]
```

方法 get_urls_in_pages 有 3 个参数，即 from_page_num、to_page_num 和 remote_ip。其中，from_page_num 和 to_page_num 与前面多线程代码中的含义一致，remote_ip 是新增的参数，是一个有两个元素的列表，第二个元素即 remote_ip[1]，表示未来当 get_urls_in_pages 作为线程运行，在抓取时所连接的远程 RPC 服务器的地址。该参数在使用 threading. Thread 生成线程时传入。该方法中的第一句

```
server = ServerProxy(remote_ip[1])
```

将根据 remote_ip[1]生成到 RPC 服务器的连接对象 server，下面的代码基本与前面章节实现 get_urls_in_pages 一致，但在 for 循环中有所变化，先前的

```
resp = requests. get(url)
```

被注释掉了，换成了

```
rstr = server. get('username',url,{},{})
```

这样抓取时就可以调用 RPC 服务器的 get 方法，即在前面 Crawler 类中定义的方法，用户名传入了 username。

在 multiple_threads_test 函数中，首先调用 server_ips = get_server_ips('remotes')，这在前面已经介绍过了，server_cnt = len(server_ips)可以获得 RPC 服务器的个数。下面介绍一下生成线程的循环体，即

```
for ind,page_range in enumerate(page_ranges_lst):
        th = threading. Thread(target = get_urls_in_pages,
                               args = (page_range[0],page_range[1],
                                       server_ips[ind % server_cnt]))
        th_lst. append(th)
```

在本例中，remotes 文本文件中恰好有 4 个 IP 地址，而 page_ranges_lst 中包含 4 组起止页，这样线程和 RPC 服务器之间就是一一对应的关系。如果不是一一对应的关系，假如 page_ranges_lst 中的起止页组数小于服务器的个数，也会正常将服务器分配到各个线程上，但服务器分配不完，会有空闲的服务器。如果 page_ranges_lst 中的起止页组数大于服务器的个数，那么 ind % server_cnt 操作的作用就显现出来了，服务器 IP 会以循环的方式依次分配到每个线程。比如，假设有 5 组起止页，有 5 个线程执行，共 4 个服务器，那么前 4 个会正常分配到服务器，第 5 个线程执行第 5 组任务，ind = 4，分配到第 4%4，即第 0 个 RPC 服务器上。

在各个云端启动 rpc_server. py 后，可在本地启动 rpc_crawler. py 进行抓取。图 4-1 给出

了 4 个云端的工作情况，在 Ubuntu 下可以安装 rdesktop，然后使用命令

```
rdesktop　　IP 地址
```

来登录远程地址。图 4-1 就是登录各个窗口后看到的情形，可以多运行 rpc_crawler. py 几次，会发现云端都会及时地响应请求，并将抓取的结果返回。

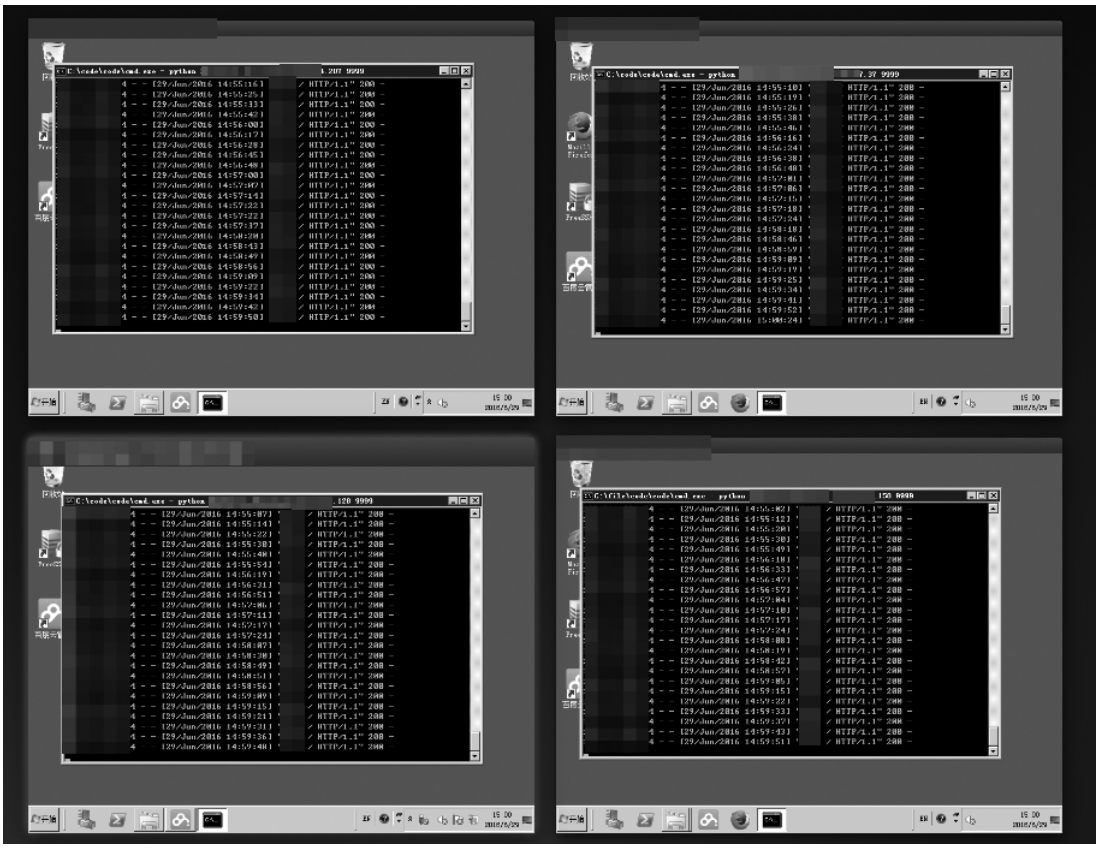


图 4-1　4 个云端 RPC 服务器的运行效果

4.2　Celery 系统

Celery 是一种分布式任务调度系统。感兴趣的读者可以登录 Celery 的官网了解 Celery 的原理、基本的使用场合与使用方法。本节不对原理详细说明，只给出利用 Celery 进行分布式抓取的实现步骤和方式。

本节给出这种方法的应用实例，在实验中仍然使用 RPC 远程调用的实例。该实例的特

点在于抓取页码范围的分组，如

```
page_ranges_lst = [
        (1,10),
        (11,20),
        (21,30),
        (31,40),
    ]
```

页码范围为 1 ~ 40，共分为 4 组，每组 10 页。在演示 Celery 分布式抓取时，我们将使用 4 台阿里云端机器来部署 Celery 的抓取任务，抓取时，只需将 4 个页码范围分别作为参数传递给 4 个云端，启动任务抓取即可。为了完成这样的功能，需要在 4 台云端上分别搭建所需的软件环境和 Python 运行时所需的模块。注意，需要安装 redis 服务器和 Python 的 Celery 模块。

4 台机器需要搭建所需的软件环境和 Python 运行时所需的模块都是相同的。以第 1 台为例，下面给出环境配置的步骤。其余 3 台配置相同，可以使用 ssh 登录 Ubuntu 云端，如图 4-2 所示。

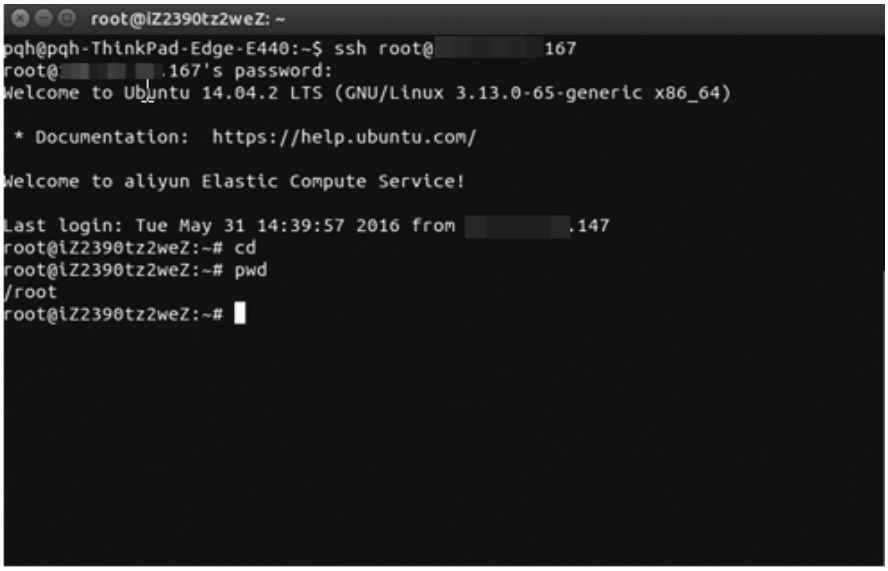


图 4-2　使用 ssh 登录 Ubuntu 云端

注意，登录时是以 root 用户登录的，因此拥有 root 权限。root 的家目录是“/root”。在 Ubuntu 环境进行软件安装时，如果是 root 用户，则可以省去 sudo。

（1）安装 redis - server

```
apt – get install redis – server
```

该命令可以安装 redis 服务器，安装后，可以运行 redis – server 启动，如图 4–3 所示。

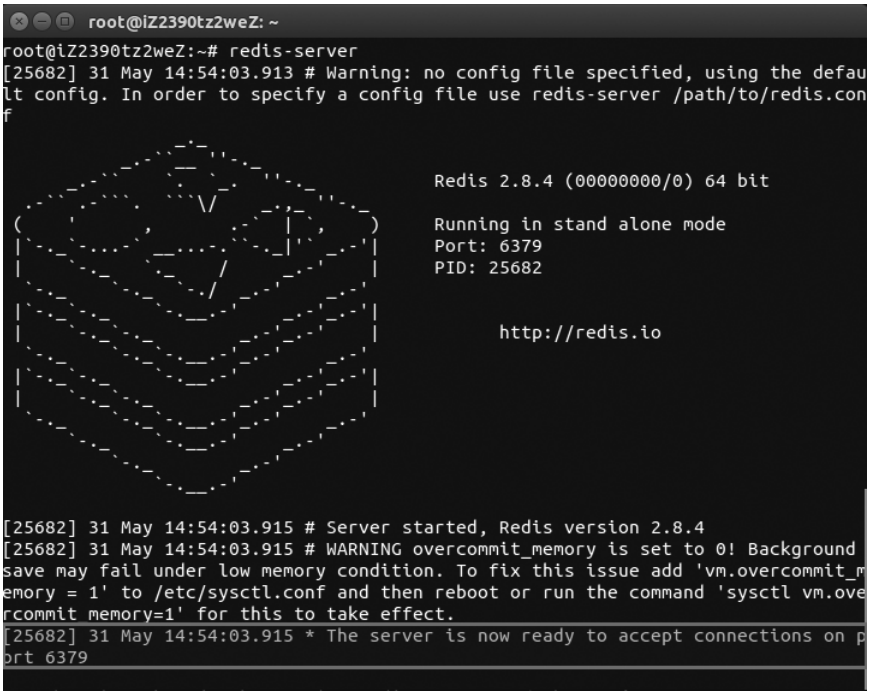


图 4–3　启动 redis – server 服务

注意，最后的矩形框表明未来该服务器将接受 6379 端口的连接这在后面 Celery 的 broker 配置中会看到。

因为在一个 shell 使用 ssh 终端登录了云端，因此启动 redis 服务后将无法进行后续的其他操作，需要先将 redis 服务关闭，再继续后面的安装操作。在这个界面下，按 Ctrl + c 即可关闭服务。

（2）安装 Celery

在云端已经安装 python2 和 python3，本例中将使用 python3，因此需将 Celery 用 pip3 来安装，但云端没有安装 pip3，可使用命令。

```
apt – get update


apt – get install python3 – pip
```

安装，然后就可以使用 pip3 安装 Celery 了，命令为

```
pip3 install celery
```

可以在 python3 的交互环境下检验是否安装成功，如图 4-4 所示。

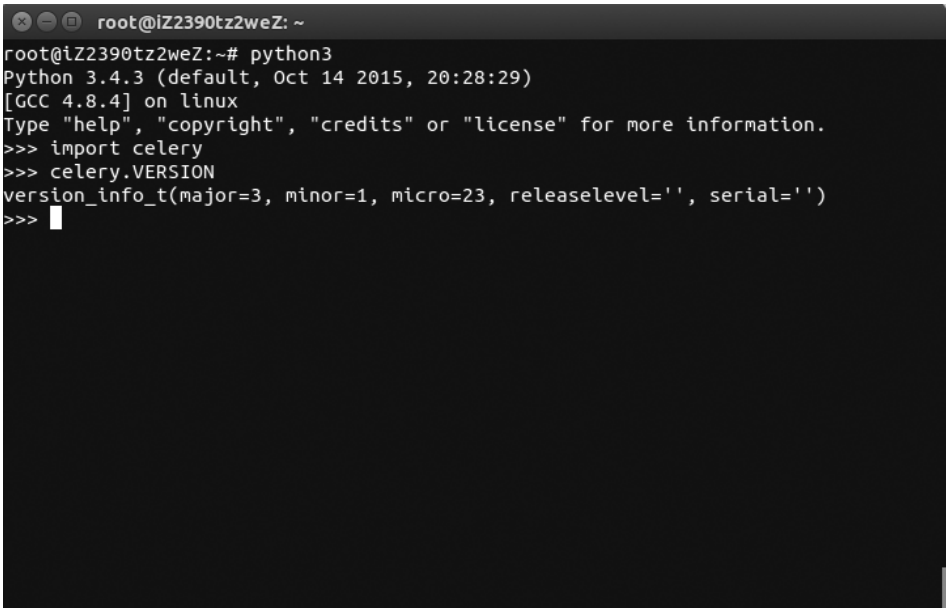


图 4-4　导入 celery 模块

（3）安装 redis

```
pip3 install redis
```

（4）安装 requests

在 Celery 的抓取任务中需要利用 requests 库，需要安装。

```
pip3 install requests
```

（5）安装 BeautifulSoup

```
pip3 install bs4
```

至此，一台云端配置完毕。其余 4 台按此配置即可。

接下来就可以编写并部署 Celery 的任务代码了。下面先给出代码，然后给出部署和启动的方法。

各云端的代码相同，代码所在的文件名为 tasks. py，内容为

```
from celery import Celery,platforms
import requests
```

```
from bs4 import BeautifulSoup

app = Celery('tasks',broker = 'redis://localhost:6379/0')

app.conf.CELERY_RESULT_BACKEND = 'redis://localhost:6379/0'

platforms.C_FORCE_ROOT = True

def format_str(s):
    return s.replace("\n","").replace(" ","").replace("\t","")

@app.task
def get_urls_in_pages(from_page_num,to_page_num):
    urls = []
    search_word = '计算机'
    url_part_1 = 'http://www.phei.com.cn/module/goods/' \
                 'searchkey.jsp?Page='
    url_part_2 = '&Page=2&searchKey='
    for i in range(from_page_num,to_page_num + 1):
        urls.append(url_part_1
                    + str(i) +
                    url_part_2 + search_word)
    all_href_list = []
    for url in urls:
        resp = requests.get(url)
        bs = BeautifulSoup(resp.text)
        a_list = bs.find_all('a')
        needed_list = []
        for a in a_list:
            if 'href' in a.attrs:
                href_val = a['href']
                title = a.text
                if 'bookid' in href_val and 'shopcar0.jsp' \
                        not in href_val and title!='':
                    if [title,href_val] not in needed_list:
                        needed_list.append([format_str(title),
                                            format_str(href_val)])
```

```
        all_href_list += needed_list
    all_href_file = open(str(from_page_num) + '_' +
                         str(to_page_num) + '_' +
                        'all_hrefs.txt', 'w')
    for href in all_href_list:
        all_href_file.write('\t'.join(href) + '\n')
    all_href_file.close()
    return len(all_href_list)
```

其中，第一句

```
app = Celery('tasks',broker='redis://localhost:6379/0')
```

生成了 Celery 实例 app，第一个参数是当前模块的名称，因为当前模块为 tasks.py，所以使用“tasks”作为参数。第二个参数是 broker 的 url，未来代码将部署在各个云端。这里使用 localhost，表示 Celery 将连接各云端本地的 redis。其中，6397 即是前面图中指出的 redis 服务端口，0 表示 redis 的默认数据库 0。下一句

```
app.conf.CELERY_RESULT_BACKEND = 'redis://localhost:6379/0'
```

给出了结果存放的位置，仍在相同 redis 中。注意，如果需要取回任务执行结果，就需要有这一句。在本例中，任务 get_urls_in_pages 有返回值，要想获得这个返回值，就需要加上这一句，在调用该任务的代码中也需要有这一句代码。

因为是以 root 身份登录云端的，所以在启动 Celery 任务时会提示不建议以 root 用户运行，如图 4-5 所示。

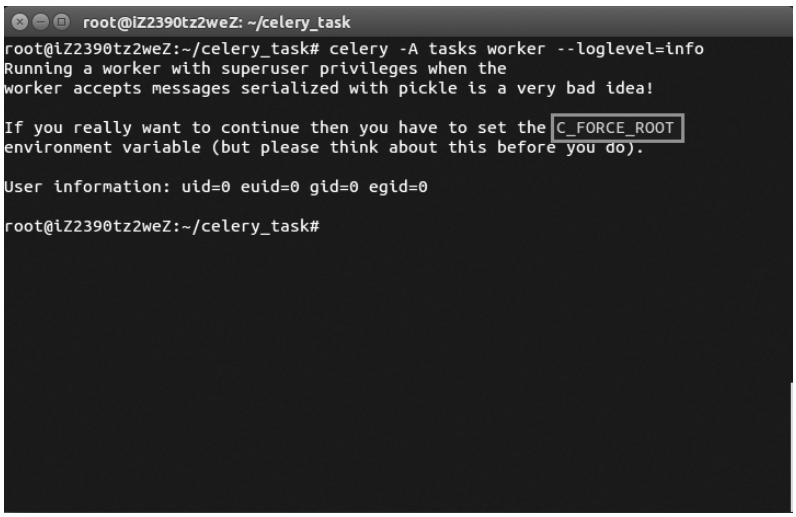


图 4-5　运行提示

最好新建一个用户来运行 Celery 任务，当前为了演示方便，也可直接使用 root 用户运行任务，根据提示可以进行以下设置，即

```
platforms. C_FORCE_ROOT = True
```

就可以以 root 身份运行代码了。

接下来的两个函数 format_str 主要用于处理字符串，get_urls_in_pages 前使用了 @ app. task 修饰器，表明未来这段代码在运行时，get_urls_in_pages 可以作为一个任务供远程的程序调用运行。get_urls_in_pages 和前面几节所定义的功能相同，都是根据参数 from_page_num 和 to_page_num 抓取指定范围内的网页并抽取出里面的 url 链接。

以上就是 Celery 执行任务的主要代码，未来将在 4 个云端部署。接下来给出抓取任务分派端的代码，在代码中主要执行的功能是将 4 组页码范围分别发送至 4 个云端，传递给相应的 Celery 任务完成工作，在代码中，为了异步地分发任务，使用了多线程技术，代码如下，所在文件为 task_dist. py。

```
from celery import Celery
from threading import Thread
import time

redis_ips = {
    0:' redis://xxx. xxx. xxx. 167:6379/0 ',
    1:' redis:// xxx. xxx. xxx. 193:6379/0 ',
    2:' redis:// xxx. xxx. xxx. 139:6379/0 ',
    3:' redis:// xxx. xxx. xxx. 7:6379/0 ',
}

def send_task_and_get_results(ind,from_page,to_page):
    print(' redis': ,redis_ips[ind])
    app = Celery(' tasks' ,broker = redis_ips[ind])
    app. conf. CELERY_RESULT_BACKEND = redis_ips[ind]
    result = app. send_task(' tasks. get_urls_in_pages' ,
                            args = (from_page,to_page))
    print(redis_ips[ind],result. get())

if __name__ == '__main__':
    t1 = time. time()
    page_ranges_lst = [
```

```
        (1,10),
        (11,20),
        (21,30),
        (31,40),
    ]

    th_lst = []
    for ind,page_range in enumerate(page_ranges_lst):
        th = Thread(target = send_task_and_get_results,
                    args = (ind,page_range[0],page_range[1]))
        th_lst.append(th)

    for th in th_lst:
        th.start()

    for th in th_lst:
        th.join()

    t2 = time.time()
    print('使用时间:',t2 - t1)
```

在这段代码中，redis_ips 是一个字典，字典的键是整数索引，值分别是 4 个云端 redis 服务器的地址。send_task_and_get_results(ind,from_page,to_page)接受 3 个参数，执行的功能是根据传入的 ind 索引值得到 redis 地址，然后像在云端一样需要执行两句

```
app = Celery('tasks',broker = redis_ips[ind])
app.conf.CELERY_RESULT_BACKEND = redis_ips[ind]
```

第一句生成一个 celery 实例，指明任务所在的模块 tasks 和该模块所在的 broker。第二句指定存储结果的 redis 数据库。

```
result = app.send_task('tasks.get_urls_in_pages',
                       args = (from_page,to_page))
```

根据生成的 app 调用 send_task 方法，将 from_page 和 to_page 参数传递给相应云端 task 模块的 get_urls_in_pages 方法，send_task 返回的对象赋给 result。get_urls_in_pages 方法返回一个整数值，表示分析出链接的个数，为了获得这个值，需要使用 result 的 get()方法。

下面给出部署各云端并启动 Celery 任务的步骤，并且给出运行本地代码并利用 Celery

任务完成抓取任务的运行结果。

1. 部署云端和启动

（1）在 4 台云端环境分别建立相关目录，并将 tasks. py 复制到相关目录中。这里分别在各云端建立/root/celery_task 目录，并将 tasks. py 复制到该目录下。

复制时可使用文件复制命令，或者使用图形化的工具，这里使用的是 pycharm IDE 下自带的 sftp 工具，可在菜单中调出，如图 4-6 所示。

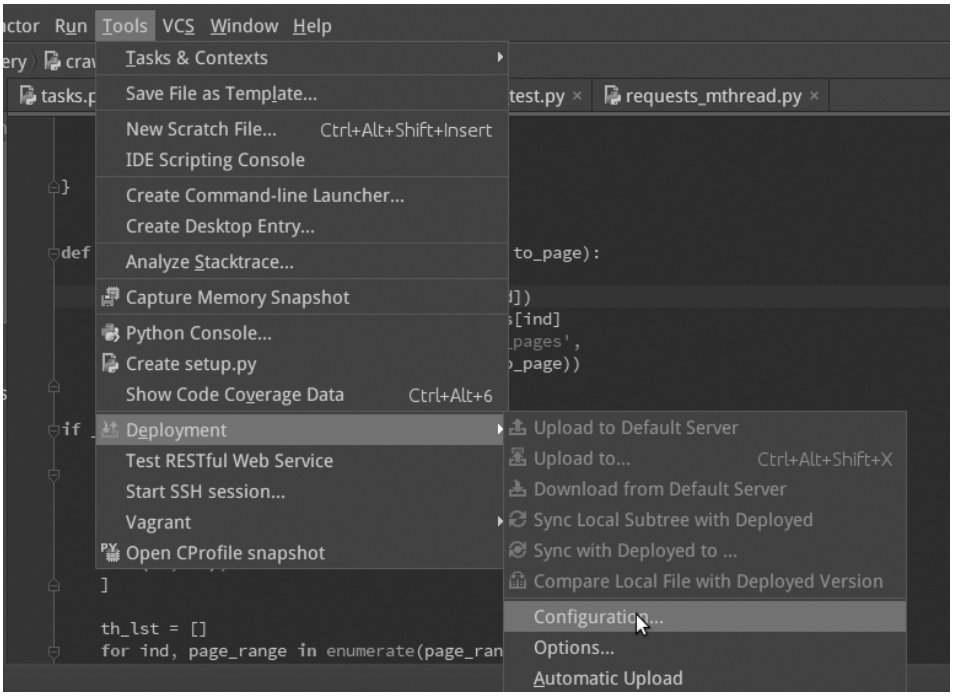


图 4-6　通过“Configuration”进入 sftp 配置界面

单击进入“Configuration”进入配置界面，如图 4-7 所示。

在配置界面中可以单击左上角的“ + ”和“ - ”选择新建或关闭配置。云端默认开启 sftp 服务，Type 可以选择“SFTP”“SFTP host”填入云端地址即可。填好并使用 ssh 登录云端时所用的用户名和密码后，可以单击“Test SFTP connection”测试一下是否可以连通。如果没有连通，则可以检查用户名和密码是否输入错误；如果连通，则单击“OK”退出。

在 IDE 的主界面右侧，单击“Remote host”即可展开远程主机的目录系统，如图 4-8 所示。这时就可以向相应的文件夹中复制文件了。如果是将同一个 IDE 的代码文件复制到远程主机，则直接拖曳即可。

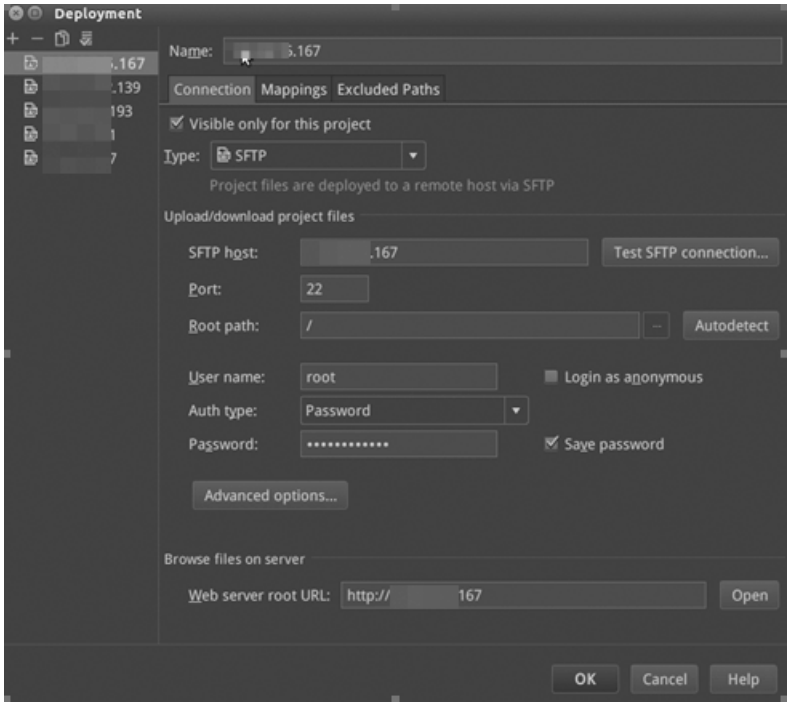


图 4-7　sftp 配置界面

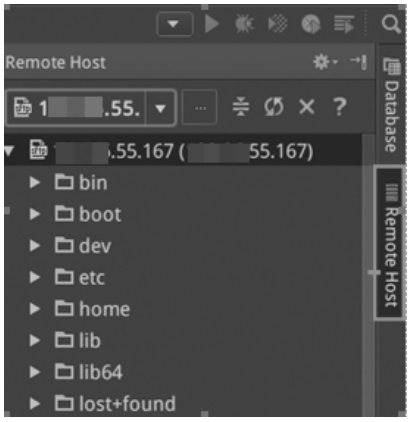


图 4-8　远程主机目录

（2）启动 Celery 任务。首先在 ssh 中使用 cd 命令进入/root/celery_ task 目录，在该目录下启动任务，命令为

```
celery -A tasks worker --loglevel=info
```

这样就可以启动 tasks 模块中的任务了。启动后，可能出现无法连接 redis 的情况，如图 4-9 所示。

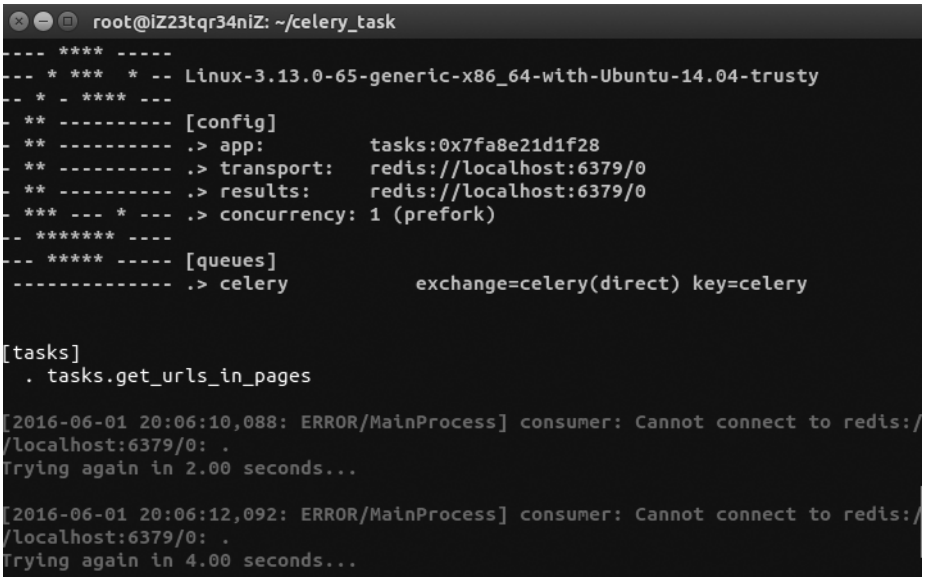


图 4-9　无法连接 redis

此时可在该 shell 中按 ctrl + c 结束 Celery 进程，然后重新启动 redis，如图 4-10 所示。

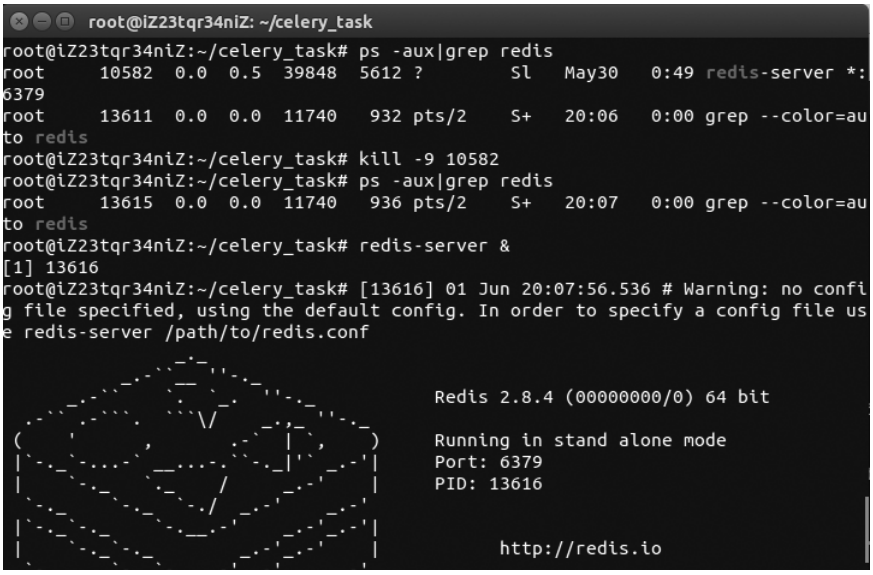


图 4-10　重新启动 redis

在命令中，ps －aux | grep redis 可以找到当前运行的 redis－server 服务，在输出结果中可知 redis－server 的 pid（进程 ID 号）为 10582。接下来使用 kill －9 10582 关闭该服务。重启 redis－server 服务，命令为 redis－server &，“&”用在 shell 命令的后面表示该命令会在后台执行。启动后，会有一些输出文字，按回车键，shell 命令提示符会再次出现，此时就

可以启动 Celery 任务了，再次运行命令

```
celery -A tasks worker --loglevel=info
```

就会看到 Celery 任务启动了，如图 4-11 所示。

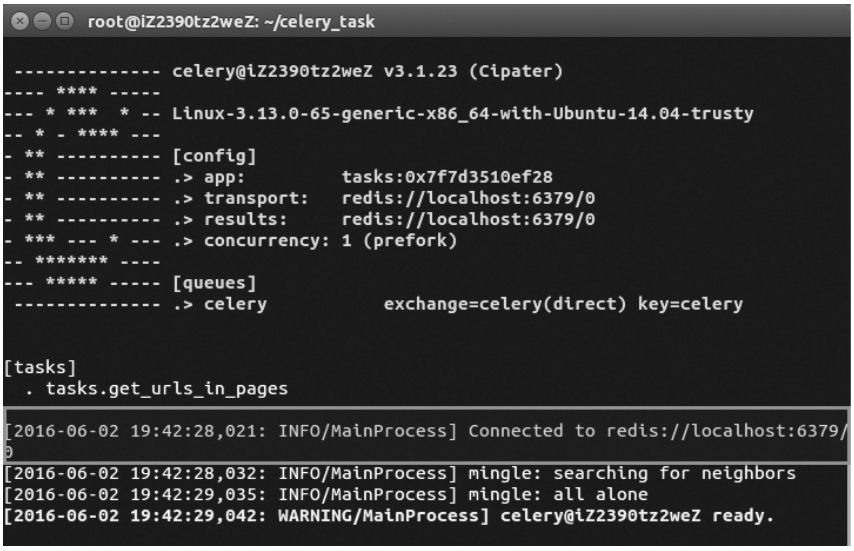


图 4-11　Celery 任务启动

2. 本地分配抓取任务

在 4 个云端分别按上述步骤启动 Celery 任务后，就可以启动 task_dist. py 脚本进行分布式抓取了。图 4-12 是抓取运行结果。

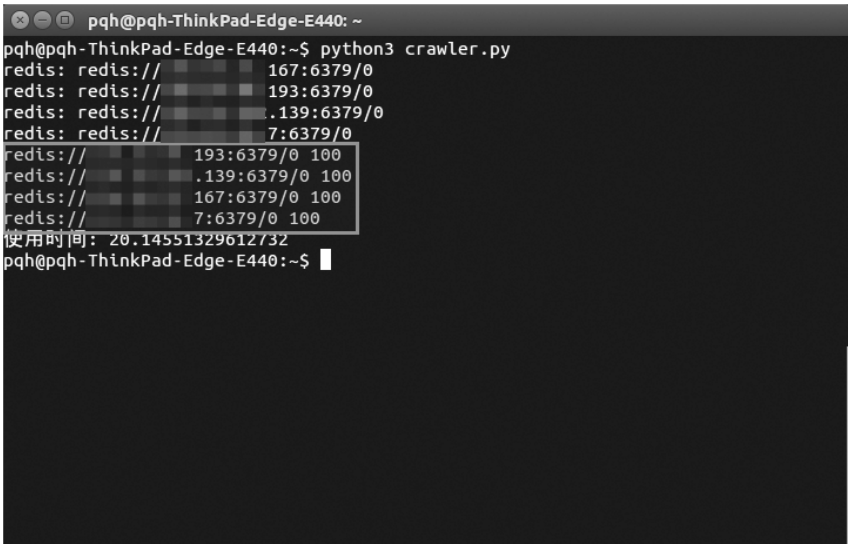


图 4-12　抓取运行结果

使用时间为 20 秒左右，完成了 4 个任务。每个任务都得到了 100 个链接，这与前面几节介绍的结果是相同的。在每个云端的 ssh 界面中可以看到 shell 的输出，以其中一个为例，如图 4-13 所示。

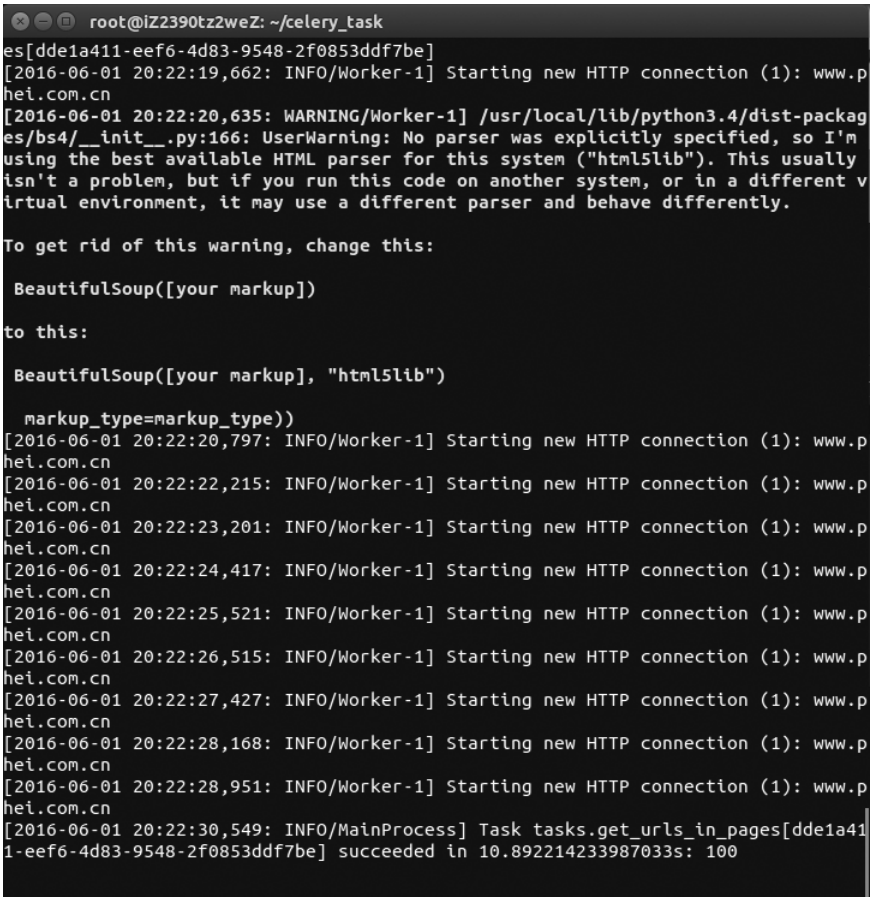


图 4-13　云端服务 shell 输出

图中共有 10 个“Starting new HTTP……”，每个都会抓取一个网页。该网页未来会抽取出 10 个图书链接。注意，第一个抓取后的提示是 BeautifulSoup 在解析网页文本时的警告信息，提示我们没有明确的指定解释器，这并不会影响分析，在后续的 9 次抓取中该信息不会再出现。每个云端都抓到了 100 个链接，可以分别到各个云端看一下这 100 个链接都是什么，这可在 IDE 中看到。图 4-14 给出了参数（11，20）页面抓取的结果，文件名称为 11_20_all_hrefs. txt，左边是文件的内容。

这里介绍的启动 Celery 服务方式有一个问题，就是在本地 shell 用 ssh 登录云端，如果用上面的方法启动 Celery 服务，就不能关闭本地启动的 shell 窗口，否则就会关闭云端启动的 Celery 服务。如果希望关闭本地 shell 窗口后启动的 Celery 任务仍在云端执行，则使用的命令

```
nohup celery -A tasks worker --loglevel=info &
```

也可将 Celery 交给 Supervisord 管理，感兴趣的读者可以查阅 Celery 的文档掌握这种方法。

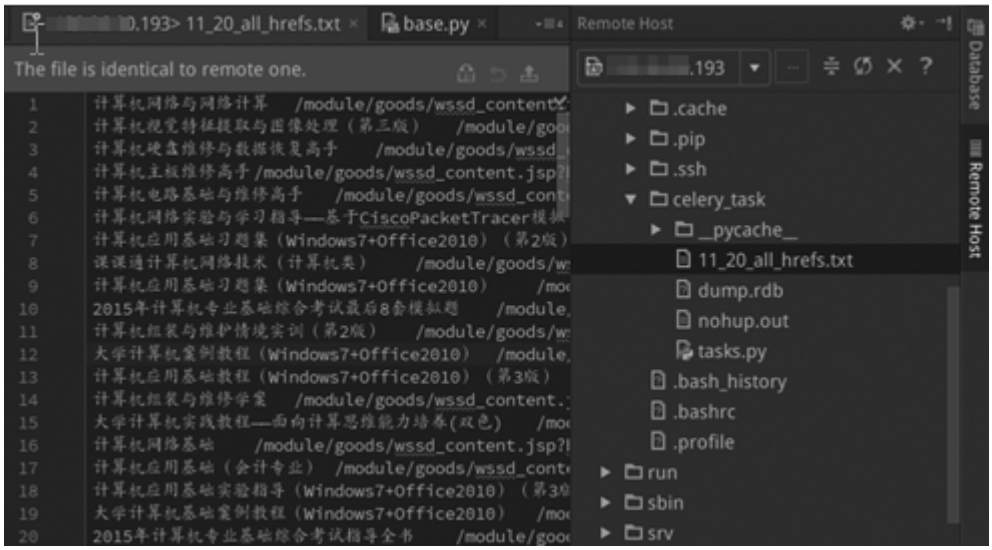


图 4-14　运行结果

实战

部署和实现 RPC 模式。如果没有云端，则可以在局域网环境或者使用虚拟机搭建一个有两个机器的局域网来进行实验。

5 第5章 全能的 Selenium

在第3章介绍单进程单机抓取时使用了 Spynner。它可以利用浏览器来完成抓取。本章介绍的 Selenium 也可以做到这一点，相比于 Spynner，Selenium 的功能更为完备，文档资料更加丰富。Selenium 主要用于 Web 开发领域的自动化测试，其提供的接口和方法也可用于数据抓取领域，基于 grid 技术可以方便地搭建分布式抓取环境。

本章主要介绍 Selenium 用于单机抓取和分布式抓取的实例。Selenium 在抓取时需要使用本地提供的浏览器，在浏览器中进行抓取，对于 Linux 非桌面系统，则无法调出浏览器运行，因此本章还介绍了如何在无图形界面的环境下使用 Selenium。

本章主要介绍将 Selenium 应用于数据抓取时所涉及的部分方法。在实际抓取时，本章介绍的内容基本可以解决绝大部分问题，因此掌握使用 Selenium 进行抓取的方式十分有意义。

5.1 Selenium 单机抓取

1. 使用 Firefox 作为实例

利用 Selenium 可以实现对多种浏览器的调用，如 Firefox、Chrome、IE、Opera 等，基本的使用流程是相似的。本小节以 Firefox 为例来说明使用 Selenium 进行网页抓取的流程。

抓取的目标仍为电子工业出版社网站。抓取首页 http://www.phei.com.cn/，代码为

```
from selenium import webdriver                    #1
import time                                       #2
firefox = webdriver.Firefox()                     #3
url = http://www.phei.com.cn/                     #4
```

```
firefox. get( url)                                    #5
time. sleep(10)                                       #6
with open( phei. html ', w ,encoding = utf8 ) as f:   #7
    f. write( firefox. page_source)                   #8
firefox. quit( )                                      #9
```

#1，从 selenium 引入 webdriver，利用 webdriver 可以初始化各种浏览器。可以使用 sudo pip3 install selenium 安装 selenium 模块。

#2，引入 time 模块，主要是为了在后面调用 sleep 函数，在页面加载后有一定的时间间隔用于观察。因为只抓取了一个页面，所以页面加载完毕后，就会继续执行下面的语句退出，可能无法看到现象。

#3，初始化一个 Firefox 浏览器用于抓取的实施。

#4，抓取的目标网页地址。

#5，调用 get 方法抓取。

#6，待#5 抓取的页面完全加载后，会有 10s 的等待时间用于观察。

#7、#8，将加载的页面保存在本地。注意，#8 中 Firefox 的 page_source 属性，存储的是页面加载后的 HTML 文本。

#9，抓取完毕后，调用 quit( )方法退出浏览器。

图 5-1 为调用 Firefox 获得的网页。

图 5-1　调用 Firefox 获得的网页

可见，被模拟的 Firefox 浏览器（见图左上角）已经被正确地初始化和调出，并且页面正确地显示了网页的内容，这与使用常规的浏览器浏览没有任何区别。注意左下角的

WebDriver 标识，如图 5-2 所示。

图 5-2　WebDriver 标识

保存的页面为一个 2300 行左右的 HTML 文本文件。

```
<html xmlns = "http://www.w3.org/1999/xhtml" > <head >
<meta content = "text/html;charset = utf - 8" http - equiv = "Content - Type" / >
<title > 电子工业出版社:科技与教育出版社,信息技术、工业技术、经济管理、大众生活和少儿科普出版商,图书、期刊、音像和网络出版商 </title >
<meta content = "IE = EmulateIE7" http - equiv = "X - UA - Compatible" / >
<meta content = "电子工业出版社:科技与教育出版社,博士后科研工作站,信息技术、工业技术、经济管理、大众生活和少儿科普出版商,图书、期刊、音像和网络出版商" name = "description" / >
<meta content = "电子工业出版社,电子社,科技与教育出版社,网上书店,经销商园地,高等教育,职业教育,信息技术,计算机,电子技术,通信,电工技术,交通,机械,工业技术,经济管理,大众生活,少儿科普出版,图书,期刊,音像,网络出版 " name = "keywords" / >
<link type = "text/css" rel = "stylesheet" href = "/templates/css/style.css" / >
<script src = "/module/publishinghouse/templates/js/diaoyongajax.js" language = "javascript" > </script >
<script language = "javascript" >
…
```

这就完成了一个网页的抓取。

注意#5 中使用的 get 方法，是 Selenium webdriver 默认支持的页面请求方式，目前还不支持 post 请求方式，但可使用 selenium - requests 第三方库来实现利用 Selenium 发送 post 请求。

2. 对 Selenium 的 profile 的配置

Selenium 在进行浏览器初始化的时候可以对浏览器的相关参数进行设置，进而控制浏览器的行为。常见的设置包括控制 CSS 的加载、控制 Javascript 文件的加载和执行、控制 images 即图片文件的下载等。

（1）控制 CSS。因为在抓取时只关心页面重要信息的获取，因此 CSS 样式表文件的加载和对页面的渲染是没有必要的，这在一定程度上可以减少抓取时间。

（2）控制 Javascript 文件的加载和执行，也可以提高抓取的效率和性能，现在网页在展

现基本信息的同时，会利用 Javascript 异步地加载很多内容。这些内容可能并不是我们的抓取目标。因此，如果我们关注的信息内容不是利用 Javascript 动态加载得到的，而且关注的信息在展示后还有大量的无关信息加载时，可以关闭浏览器 Javascript 的加载和执行功能，将会极大地提高抓取的性能。

如果我们希望获得的内容是利用 Javascript 动态加载的，那么就不能阻止 Javascript 的加载与执行了。此时有这样一种可能，Javascript 加载了我们关注的信息之后，还会加载很多无关的信息，这样也会影响抓取的性能。在默认的情况下，Selenium 控制的浏览器会像传统的浏览器一样，会加载完所有应展示的内容。因此，抓取时按页进行逐页抓取也是如此，一页一页地加载大量无关内容。这种情况可利用 Selenium 提供的机制进行实时观察，当出现所关注的信息后即停止页面后续内容的处理和加载，可极大地提高抓取效率。

（3）控制图片的加载。与以上的描述类似，控制图片的加载也可以极大地提高抓取效率。一张图片的大小可能远远超过网页需要加载的 CSS 文件和 Javascript 脚本文件的大小，并且很多网站的图片较多，由此会造成极大的带宽开销并使抓取效率降低。控制图片的加载可以通过阻止图片的加载来解决这个问题。需要注意的是，Selenium 控制的浏览器即使展示出图片，也无法将其保存在本地。如果希望抓取图片，则需要其他的库，如 requests 的协助。后面将会给出这样的实例。

下面用实例说明如何在利用 Selenium 抓取时做到以上几点，仍以 Firefox 作为实例。在初始化 Firefox 时，对使用偏好进行相应的设置即可达到目标。其实使用常规的 Firefox 浏览器上网浏览时，也可以通过设置相应的参数来改变浏览器的行为。打开 Firefox，在网站地址输入栏输入“about:config”，回车，如图 5-3 所示。

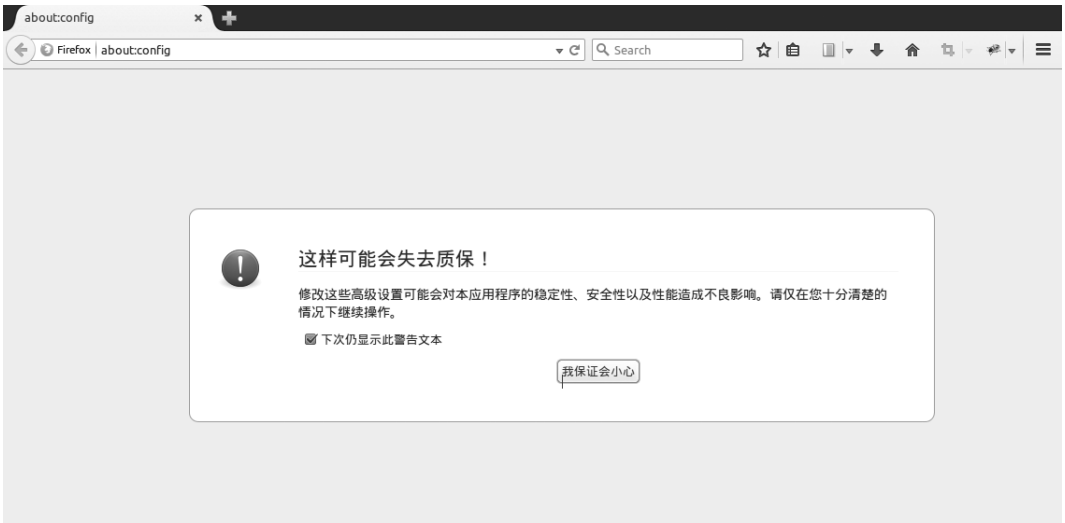


图 5-3　about:config 页面

单击“我保证会小心”，出现如图 5-4 所示的界面。

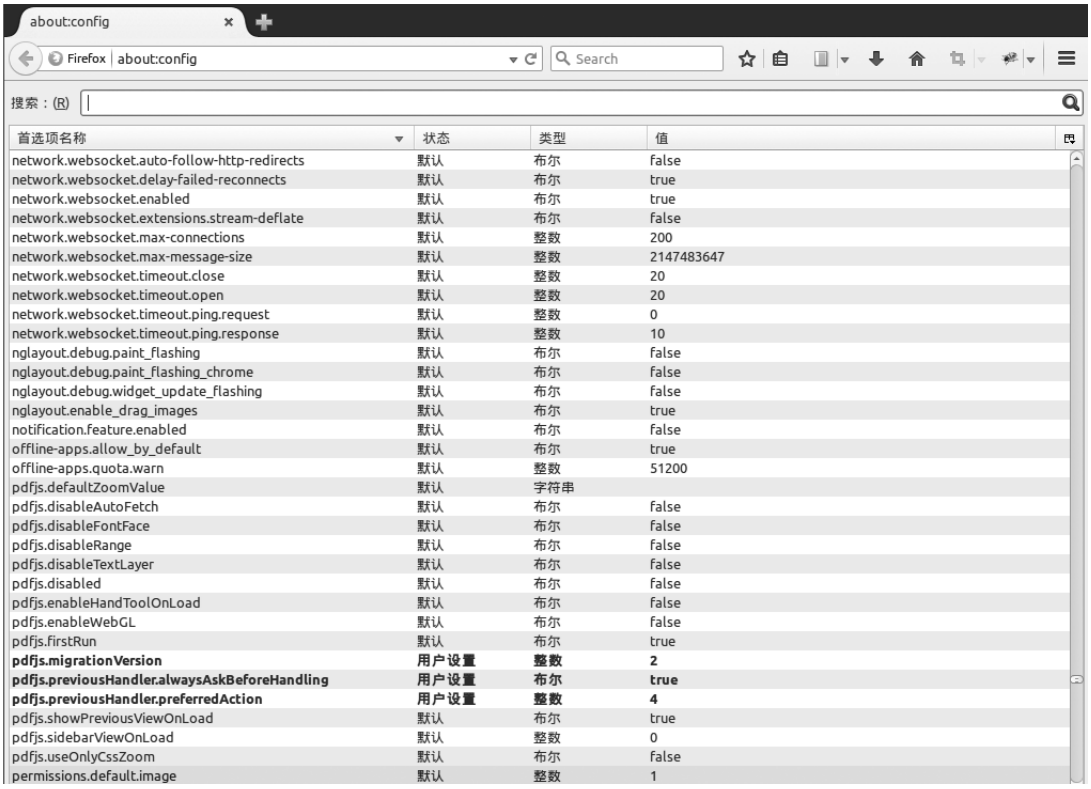


图 5-4　首选项信息界面

在这个界面中最左列的“首选项名称”就是可以设置的浏览器相关参数，通过对这些参数进行设置可以改变浏览器的行为。比如，希望浏览器在打开网页时不下载和加载图片文件，那么设置首选项名称为“permissions. default. image”的值即可。首选项名称列表上方的搜索框中输入“permissions. default. image”即可定位到该项，如图 5-5 所示。

图 5-5　permissions. default. image 选项信息

第一项就是定位到的首选项，后面给出了该项的“状态”“类型”和“值”。当前的值为 1，双击该项，在弹出的对话框中将值设置为 2，如图 5-6 所示。

图 5-6　permissions. default. image 值设置

单击“确定”，这时就已经修改浏览器的行为了，再打开电子工业出版社首页“http://www. phei. com. cn/”，如图 5-7 所示。

图 5-7　首页不显示图片

可以看到，所有的图片位置均未显示，说明设置起到了作用，注意中上部显示的不是图片，而是 flash。

以上就是通过修改首选项参数来改变浏览器行为的方法，注意使用限制图片加载的方法后将参数恢复为 1，否则，以后打开网站时均不会显示图片。

上面是手工设置的过程，Selenium 提供了可以通过编程设置相关参数以改变浏览器行为的方法，通过在初始化时载入参数配置文件达到控制浏览器行为的效果。参数的配置是通过初始化 FirefoxProfile 文件，然后进行相关设置，再用配置好的文件作为参数对 Firefox 浏览器进行初始化，这时的 Firefox 就具有了相应参数指定的行为。下面的代码给出了限制浏览器加载 CSS、禁止执行 Javascript、禁止图片下载和显示的方法。

```
from selenium import webdriver
import time
firefox_profile = webdriver. FirefoxProfile( )                          #1
firefox_profile. set_preference( "permissions. default. stylesheet" ,2)  #2
firefox_profile. set_preference( "permissions. default. image" ,2)       #3
firefox_profile. set_preference( "javascript. enabled" ,False)           #4
firefox_profile. update_preferences( )                                  #5
firefox = webdriver. Firefox( firefox_profile)                          #6
url = http://www. phei. com. cn/
print( start load )
t_start = time. time( )
firefox. get( url)
t_end = time. time( )
print( loading time is: ,t_end - t_start)
print( ++++++++++++++ )
time. sleep(10)
firefox. quit( )
```

其中，

#1，初始化一个 FirefoxProfile 实例 firefox_profile；

#2，禁用样式表文件；

#3，不下载和加载图片；

#4，禁止 Javascript 的执行；

#5，更新设置；

#6，利用 firefox_profile 初始化浏览器，此时浏览器初始化 Firefox 具有参数所设定的行为。

运行这段代码后得到的页面如图 5-8 所示。

注意，关闭禁止 Javascript 的执行后，flash 也相应地消失了。

命令行的输出为

```
start load
loading time is: 0. 3912169933319092
++++++++++++++
```

其中，0. 3912169933319092 秒是从浏览器发出请求至所需页面内容加载完成的时间间隔。

图 5-8　关闭各选项后的情况

当#4 行的参数由 False 改为 True 后再运行这段代码，会发现 flash 出现了，如图 5-9 所示。

图 5-9　Javascript 选项开启对页面的影响

此时的命令行输出为

```
start load
loading time is: 2.473635196685791
++++++++++++++
```

这时的加载时间要比禁用 Javascript 时要多。无论是否禁用 Javascript，页面加载完成的时间都与程序一次运行的实际网络环境有关。如果确实需要比较这两种方式，则可以在每种方式下运行相同的次数，然后取平均值进行对比。

下面给出这种对比方法。扩展一下，即不只对比是否禁用 Javascript，把 CSS 和图片的加载与否也考虑进来的代码如下。其中，绘图使用了 matplotlib，也可以使用下面的命令安装，即

```
sudo apt-get install python3-matplotlib
```

代码为

```
from selenium import webdriver
import time
import matplotlib.pyplot as plt                                    #1
def performance(n,css_val,image_val,js_flag):                      #2
    loading_time = []
    for i in range(0,n):
        firefox_profile = webdriver.FirefoxProfile()
        firefox_profile.set_preference("permissions.default.stylesheet",
                                       css_val)
        firefox_profile.set_preference("permissions.default.image",
                                       image_val)
        firefox_profile.set_preference("javascript.enabled",
                                       js_flag)
        firefox_profile.update_preferences()
        firefox = webdriver.Firefox(firefox_profile)               #3
        url = 'http://www.phei.com.cn/'
        print('start load')
        t_start = time.time()
        firefox.get(url)
        t_end = time.time()
        print('loading time is:',t_end-t_start)
        loading_time.append(t_end-t_start)
        print('++++++++++++++')
```

```
            # time. sleep(10)
            firefox. quit( )                                    #4

        return [ x for x in range(1,n + 1) ],loading_time       #5

if __name__ == __main__ :

        x1_lst,y1_lst = performance(10,1,1,True)                #6
        x2_lst,y2_lst = performance(10,2,2,False)               #7

        ava_y1 = sum(y1_lst)/len(x1_lst)                        #8
        ava_y2 = sum(y2_lst)/len(x2_lst)                        #9

        plt. title("Compare loading time. ")                    #10
        plt. xlabel("Test number")                              #11
        plt. ylabel("Loading time")                             #12
        plt. plot(x1_lst,y1_lst', go': ,label = str(ava_y1))    #13
        plt. plot(x2_lst,y2_lst', rs': ,label = str(ava_y2))    #14
        plt. legend( )                                          #15
        plt. show( )                                            #16
```

#1，引入 matplotlib 绘图库。

#2，定义性能测试函数，该函数接受 4 个参数：

n：实验次数；

css_val：数据类型为整型，1 代表正常处理 css，2 代表禁用 css；

image_val：数据类型为整型，1 代表正常处理图片，2 代表禁用图片；

js_flag：数据类型为布尔型，True 代表正常处理 Javascript 脚本，2 代表禁用 Javascript 脚本。

#3，利用参数配置文件初始化 Firefox 浏览器。

#4，关闭#3 中初始化的浏览器。注意，#3 和#4 是在一次循环中成对出现的，也就是 n 次实验中的每一次都需要打开和关闭浏览器，或者说，n 次实验是彼此独立的。

#5，返回两个列表。第一个列表其实就是 $[1,2,\cdots n]$，第二个列表中的元素依次分别对应每次实验页面加载完毕的时间，可记为 $[t_1,t_2,t_3,\cdots,t_n]$。在绘图时，前者就是横轴的数据，后者作为纵轴的数据。

#6，进行实验 1，实验次数为 10 次，CSS、图片和 Javascript 正常处理。

#7，进行实验 2，实验次数为 10 次，CSS、图片和 Javascript 全部禁用。

#8，计算实验 1 的平均加载完成时间。

#9，计算实验 2 的平均加载完成时间。

#10 ~ #16，给出了 matplotlib 绘图基本方式和函数。其中，

#10，设置图片的标题；

#11，设置横轴，即 x 轴代表的含义；

#12，设置纵轴，即 y 轴代表的含义；

#13，绘制实验 1 的图线。x1_lst、y1_lst 分别是 x 轴和 y 轴的数据；' go：' 是图线的展示方式；' g 代表 green，即图线结点的颜色；' o 是结点的形状，为圆形；' :' 是结点间的连接方式，是以虚线连接的；label 给出图线的标识，这里使用平均加载完成时间作为图线的标识。

#14，绘制实验 2 的图线。x2_lst、y2_lst 分别是 x 轴和 y 轴的数据；' rs：' 是图线的展示方式；' r 代表 red，即图线结点的颜色；' s 是结点的形状，为方形；' :' 是结点间的连接方式，是以虚线连接的；lable 含义同上。

#15，在#13 和#14 中设置的 label 标识，需要运行此句才能正确显示。

#16，显示绘制的图片。

运行这段代码后显示的图片如图 5-10 所示，结果可能因为网络延时等原因有所差异。

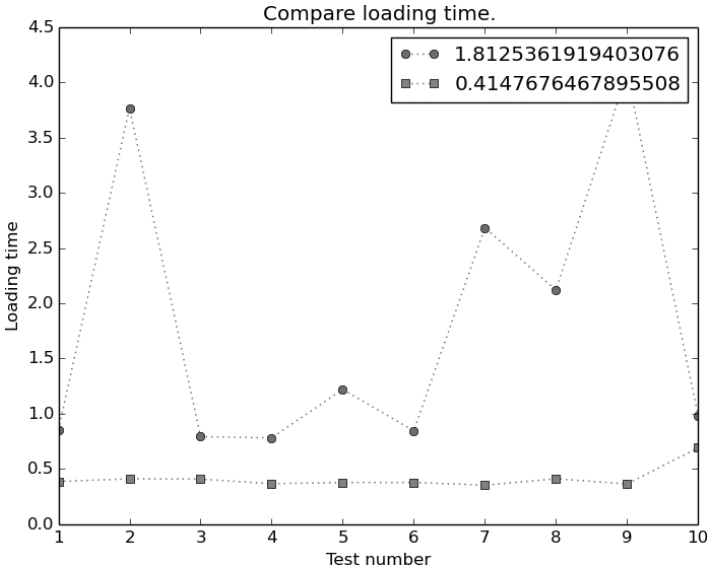


图 5-10　选项的设置对加载时间的影响

从如图 5-10 所示中可以看出：

（1）如果没有限制 CSS、图片和 Javascript 运行方式（圆点连接的虚线）的平均时间为 1.8 秒左右，那么限制三者后（方块连接的虚线），平均页面加载时间只有 0.4 秒左右。

（2）对三者进行限制的曲线变化比较平缓，这是因为无需进行 Javascript 的下载和运行（Javascript 运行时也会同步或异步地发出网络请求，对相关数据请求和加载），不用下载图片，并且无需进行 CSS 渲染，只是返回文本信息，对网络依赖较少；无限制的版本在下载图片、运行 Javascript 并进行网络请求时会有很多网络交互，对网络环境依赖较大，所以每次加载时间可能会有较大的差异。

一般在抓取时目标都是一些文本值，因此对浏览器的参数进行一些设定可以有效地提高抓取效率。比如，在我们希望抓取首页上的图书分类中，通过对科技图书所包含的各子类名称的分析可以发现（也可以结合前面的两个图分析出来），它们都是文本值并且不依赖于 Javascript 的动态加载，因此这时关闭 Javascript，禁止 CSS 和图片的下载与加载，可以提高速度和效率。代码为

```
from selenium import webdriver
import time
from bs4 import BeautifulSoup                                    #1
def get_info(html):                                              #2
    bs = BeautifulSoup(html)
    tables_lst = bs.find_all('table',attrs={'class':'tsflxin'})
    a_lst = tables_lst[0].find_all('a')
    for a in a_lst:
        print(a.text)
firefox_profile = webdriver.FirefoxProfile()
firefox_profile.set_preference("permissions.default.stylesheet",2)
firefox_profile.set_preference("permissions.default.image",2)
firefox_profile.set_preference("javascript.enabled",True)
firefox_profile.update_preferences()
firefox = webdriver.Firefox(firefox_profile)
url = 'http://www.phei.com.cn/'
print('start load')
t_start = time.time()
firefox.get(url)
t_end = time.time()
print('loading time is:',t_end-t_start)
print('++++++++++++++')
get_info(firefox.page_source)                                    #3
time.sleep(10)
firefox.quit()
```

其中，

#1，引入 BeautifulSoup 作为分析库。

#2，定义 get_info 函数，分析参数 HTML 文本，从中提取出科技图书的各子类名称。

#3，调用 get_info 函数，将 firefox. page_source 作为参数传入。

运行这段代码的结果为

```
start load
loading time is: 0.46660947799682617
+++++++++++++++

计算机
电子技术
通信与网络
电工技术
机械/仪器仪表
材料科学与工程
汽车
交通
新能源(能源与动力工程)
建筑
军事
考试与认证_科技
其他
```

当面对大量抓取，并且对时间要求比较紧迫时，使用这种方式可以极大地加快速度，减少带宽。

3. 下载图片

下载图片是抓取任务中常见的环节。虽然 Selenium 可以模拟浏览器，在浏览器中显示图片，但并不具备图片下载的功能函数。可以利用其他的库，如 requests 弥补这个问题。使用 requests 下载图片的代码为

```
import requests                                                    #1
url = http://www.phei.com.cn/templates/images/logo.jpg            #2
r = requests.get(url) #3
with open( phei_log.jpg ', wb ) as f:                              #4
    f.write(r.content)                                             #5
```

#1，引入 requests。

#2，要抓取的图片 url。

#3，利用 requests 访问 url，访问返回的响应存在 r 中。

#4～#5，将返回的图片保存在本地。其中，

#4，保存的文件名为“phie_log. jpg”，参数“wb”表示以二进制方式（b）写入（w）。

#5，写入图片，注意返回的图片是二进制的形式，r. content 表示以二进制的方式读取响应内容。

运行上面的代码。运行成功后，在代码所在文件夹中将出现“phie_log. jpg”文件，打开后如图 5-11 所示。

图 5-11　下载后的图片

4. 电商网站动态内容抓取

很多电商网站首页的内容是动态加载的，使用 requests 是无法抓取全部首页内容的，首页的内容并不是一次加载完的，而是随着用户向下浏览而逐步展现的，这样又为内容的获取增加了难度。Selenium 不但可以抓取到动态加载的内容，也提供了针对浏览器的各类操作。这些操作可以模拟常见的鼠标、键盘操作，执行 Javascript 脚本可达到控制浏览器的目的。本节以京东商城首页为例，介绍在动态内容获取的基础之上，配合 Selenium 提供的浏览器操作方法进行抓取的实例。

首先看一下使用 requests 抓取的情况，下面是抓取代码，保存在文件 requests_jd. py 中。

```
import requests
url = https://www. jd. com/
resp = requests. get(url)
s = resp. text
print(len(s))
with open( requests_jd. html', w ,encoding = gbk ) as f:
    f. write(s)
```

代码首先引入 requests 模块，url 为抓取地址。本例使用 get 方法进行抓取，返回的响应对象置于 resp 中，利用响应对象的 text 属性可以获得响应的文本内容。本例中即为 HTML 的文本内容。print(len(s))给出了文本的长度，通过长度可以估计抓取的内容。一般长度

越长，说明抓取的文本内容越多，抓取得就越完全。最后将抓取的内容以 gbk 形式编码，保存在文件 requests_jd. html 中。这就是上述代码需要完成的工作。下面运行这段代码，结果如图 5-12 所示。

```
pqh@pqh-ThinkPad-Edge-E440:~$ python3 requests_jd.py
216714
```

图 5-12　requests 获得的首页代码长度

图中的数字表示抓取 HTML 文本内容的长度为 216714 字节（有可能随着网页内容有所变化）。第一次抓取会得到 1253 字节，在随后的运行中会得到 216714 字节，这可能与服务端的验证有关，不过 216714 字节是本例中 requests 所能抓到的全部内容。

保存的 requests_jd. html 文件为普通的 HTML 文件，可以使用浏览器打开查看内容，下面是使用 Firefox 查看结果，如图 5-13 所示。

图 5-13　使用 Firefox 查看的结果

当下拉滚动条时会出现如图 5-14 所示的界面。

这说明并没有抓到全部的内容。在真实环境下浏览网页时，也会发现网页的内容不是一次性完全加载完成的，随着浏览网页内容会被逐步加载，利用鼠标的滚轮或拖动右侧的滚动条向下滚动页面时，页面的内容会随着滚动逐渐加载。根据这一特点，可以使用 Selenium 来完成抓取工作。下面的实例利用 Selenium 执行 Javascript 脚本的能力，利用脚本控制浏览器的滚动条，并通过滚动条的滚动动态加载内容。代码如下，保存在文件 selenium_jd. py 中。

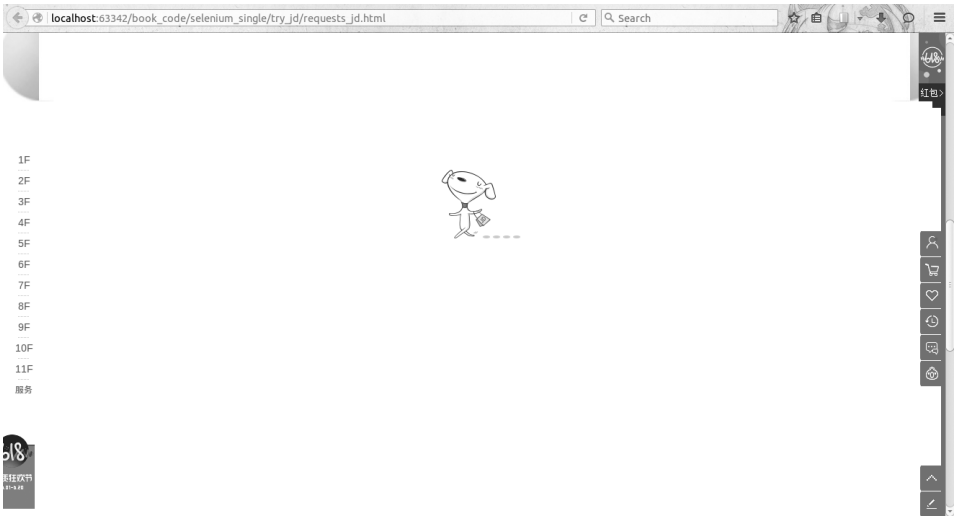


图 5-14　下拉滚动条后出现的界面

```
from selenium import webdriver
import time

def scroll(n,i):
    return "window.scrollTo(0,(document.body.scrollHeight/{0})*{1});".\
            format(n,i)

url = 'https://www.jd.com/'
firefox = webdriver.Firefox()
firefox.maximize_window()
firefox.get(url)

n = 10
for i in range(1,n+1):
    s = scroll(n,i)
    print(s)
    firefox.execute_script(s)
    time.sleep(2)

print(len(firefox.page_source))
with open('selenium_jd.html','w',encoding='gbk',errors='ignore') as f:
    f.write(firefox.page_source)
```

首先，从 selenium 引入 webdriver，再引入 time。其中，webdriver 用于初始化 Firefox 浏览器，引入 time 的原因在于将在滚动条向下滚动的过程中，每滚动一次，调用 time 的 sleep 方法使脚本暂停一定的时间间隔，以保证因滚动而加载的动态内容加载完成。

接下来的函数 scroll(n,i)可将返回组合进参数 n 和 i 的字符串。该字符串是一个浏览器可执行的 Javascript 脚本。脚本中 window. scrollTo(x,y)可以将显示的内容滚动到指定的坐标。其中，x 和 y 分别表示要在窗口文档显示区左上角显示文档的 x 坐标和 y 坐标。因此，根据这个定义，在滚动时保持 x 为 0，y 不断增加即可。这样在浏览器中就可以模拟页面向下滚动的效果。document. body. scrollHeight 可以获得滚动内容的高度，未来在调用 scroll 函数时传入的参数 n 是固定不变的，而 i 会动态地增加，每次加 1，这样

```
(document. body. scrollHeight/{0}) * {1}
```

的效果就是随着 i 的变化，字符串的形式也随着变化，对应的结果数值也随着增加变化，因为结果数值又作为 window. scrollTo(x,y)方法的 y 值，因此可以达到滚动的效果。下面的 4 句就是指定 url，初始化 Firefox 浏览器，然后最大化浏览器，接着就是调用

```
firefox. get(url)
```

进行抓取了。

```
n = 10
for i in range(1,n + 1):
    s = scroll(n,i)
    print(s)
    firefox. execute_script(s)
    time. sleep(2)
```

在这里将 n 设置为 10，n 值可以根据实际情况设置。在 for 循环中，i 的值可以由 1 取到 n。在循环体中，s = scroll(n,i)的作用是调用 scroll 函数将 n 和 i 的值传入，根据传入的值 scroll，会将它们组合进所返回的字符串表达式，返回的结果赋给变量 s，如调用 scroll(10,1)，返回的结果 s 为

```
window. scrollTo(0,(document. body. scrollHeight/10) * 1);
```

因为 s 中包含的是 Javascript 脚本，可以调用

```
firefox. execute_script(s)
```

来执行这段脚本。time. sleep(2)的作用是在执行脚本使内容滚动后，等待一段时间，待内

容加载完成后，再进行下一次滚动。

最后输出本次抓取的内容长度，并将结果保存在 selenium_jd. html 文件中，因为在实际运行时发现保存文件的编码出现了错误，因此在文件函数中加入了 errors = 'ignore' 参数，表示忽略出现的编码错误。

运行命令和结果如图 5-15 所示。

```
pqh@pqh-ThinkPad-Edge-E440:~$ python3 selen_async_bak.py
window.scrollTo(0, (document.body.scrollHeight/10)*1);
window.scrollTo(0, (document.body.scrollHeight/10)*2);
window.scrollTo(0, (document.body.scrollHeight/10)*3);
window.scrollTo(0, (document.body.scrollHeight/10)*4);
window.scrollTo(0, (document.body.scrollHeight/10)*5);
window.scrollTo(0, (document.body.scrollHeight/10)*6);
window.scrollTo(0, (document.body.scrollHeight/10)*7);
window.scrollTo(0, (document.body.scrollHeight/10)*8);
window.scrollTo(0, (document.body.scrollHeight/10)*9);
window.scrollTo(0, (document.body.scrollHeight/10)*10);
601669
pqh@pqh-ThinkPad-Edge-E440:~$
```

图 5-15　运行命令和结果

命令行的结果显示了每一次滚动时执行的 Javascript 脚本，最后还给出了抓取的内容长度为 601669 个字节，远大于前面 requests 获得的字节数。图 5-16、图 5-17 和图 5-18 给出了抓取过程前几次执行脚本时滚动条的变化情况，注意观察右侧滚动条的位置。

图 5-16　抓取时滚动条的变化 1

图 5-17　抓取时滚动条的变化 2

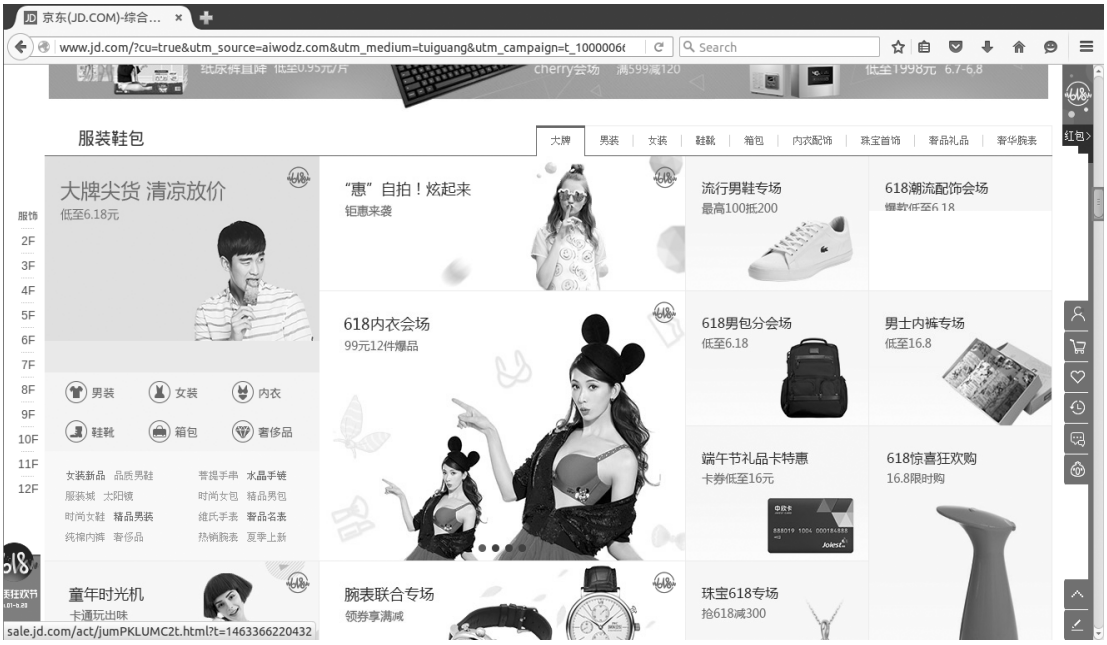


图 5-18　抓取时滚动条的变化 3

最后，打开保存的 selenium_jd. html 文件，可以看到里面显示的内容要多于requests 抓取的内容。

5.2 Selenium 分布式抓取

使用 Selenium 可以进行分布式抓取，在前面的实例中使用了远程过程调用（RPC）技术，结合 requests 模块实现了一种类似于代理的分布式抓取方案。在该方案中因为使用的是 requests 模块，因此对于网页的动态内容是无法捕获到的。本节介绍的 Selenium 模块可以对动态内容加载并解析，如果能够将其应用在分布式抓取环境，也能提高抓取的效率。Selenium 库自身实际上就提供了相似的功能。本节介绍如何利用这个功能搭建一个简单的分布式抓取环境。

这里使用 Selenium - grid 2.0 提供的主要技术，对于 Selenium - grid 2.0 主要特点感兴趣的读者可以查看相应的官方文档。一个 grid 由一个 hub 和一个或多个 nodes 组成。本节将使用 1 个 hub 和 4 个 nodes 完成分布式抓取工作。

实例中使用 4 个负责抓取的云端作为 nodes，1 个云端作为 hub。云端采用的是阿里云，都具有固定的 IP，服务器根据需要选择即可。5 台云端的配置为：1G 内存，单核 CPU，40G 硬盘，与 RPC 分布式系统具有相同的硬件配置，如图 5-19 所示。

图 5-19　云端节点参数

该分布式结构的基本工作方式是 hub 云端与 4 个 nodes 云端相连接，在 hub 和各 nodes 上分别部署和启动相应的 Selenium 服务程序，然后在本地编写代码，将需要抓取的链接发送至各个 nodes。nodes 利用 Selenium 的 webdriver 抓取网页，并将网页返回后，就可以根据需要对网页内容进行相应的处理了。各云端的 IP 地址和节点类型见表 5-1。

下面给出完成上述功能所需的准备工作，需要在 hub 和各 nodes 上安装和下载以下软件。

（1）需要安装 java

首先需要在云端安装 jdk。因为实例中使用云端处理器和内存的限制，所以建议安装 jdk1.6 版本，安装 jdk7 或 jdk8 可能无法安装成功。

（2）Firefox 和 Selenium 的选择

在实例中使用 Firefox 浏览器进行抓取。Selenium 的不同版本对应的 Firefox 版本也有所差异。表 5-2 给出了版本的对应关系。

表 5-1　各云端的 IP 地址和节点类型

IP	节点类型
xxx. xxx. xxx. 158	hub
xxx. xxx. xxx. 102	node
xxx. xxx. xxx. 38	node
xxx. xxx. xxx. 105	node
xxx. xxx. xxx. 184	node

表 5-2　云端 Selenium 和 Firefox 版本的对应关系

Selenium	FireFox
2. 25. 0	18
2. 30. 0	19
2. 31. 0	20
2. 42. 2	29
2. 44. 0	33

在实例中将使用 Selenium - server - standalone - 2. 44. 0. jar，因此需要安装 Firefox 33 版本。

（3）下载 Selenium - server - standalone - 2. 44. 0. jar

Selenium - server - standalone - 2. 44. 0. jar 可在网上下载，下载后可保存在相应的目录下，在需要时可在该目录中运行。

下面给出具体的实例步骤。

在前面进行了软件安装和下载后，就可以在各云端启动相应的命令使它们行使各自的职责。通常需要首先启动 hub，因为所有的 nodes 都依赖于 hub，但先启动 nodes 也无妨，因为在启动 hub 后，nodes 会自动识别出 hub 并连接上。这里先启动 hub，然后启动各 nodes。

首先登录 hub 节点 ×××. ×××. ×××. 158，在 Ubuntu14. 04 操作系统中，可以利用 rdesktop 登录远程 Windows 桌面系统。安装也很简单，再一次给出命令

```
sudo apt - get install rdesktop
```

安装后，可在 shell 中使用命令 rdesktop 启动远程桌面，如图 5-20 所示。

启动后即可使用相应的用户名和密码登录 Windows 远程桌面。登录后进入 selenium - server - standalone - 2. 44. 0. jar 包所在的文件夹，调出 Windwos 的 cmd 并导航到该文件夹，输入命令 java　- jar selenium - server - standalone - 2. 44. 0. jar　- role hub，如图 5-21 所示。

这样在 ×××. ×××. ×××. 158 就成功启动了 hub，接下来需要启动其他 4 个 nodes。启动 nodes 的方式和启动 hub 的方式类似，差别在于命令的形式。这里以 ×××. ×××. ×××. 102 节点为例，在通过 rdesktop 连接成功并进入系统之后，利用下面的命令启动 node，即

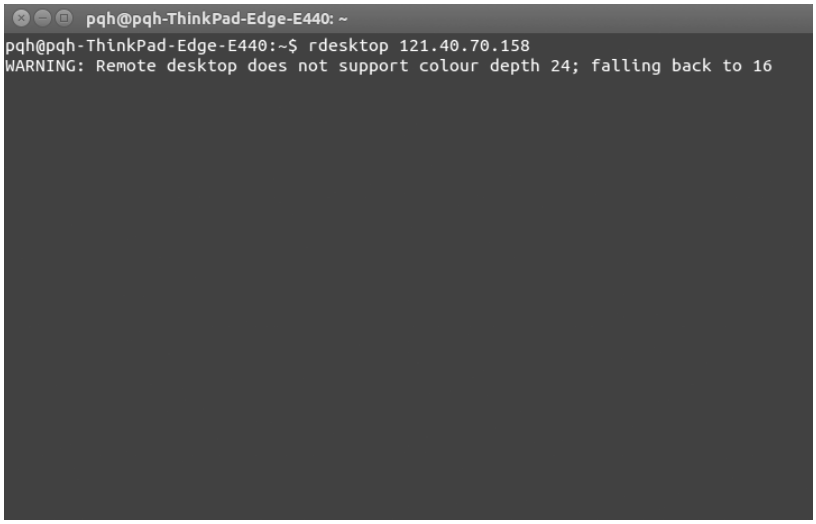


图 5-20　使用 rdesktop 启动远程桌面

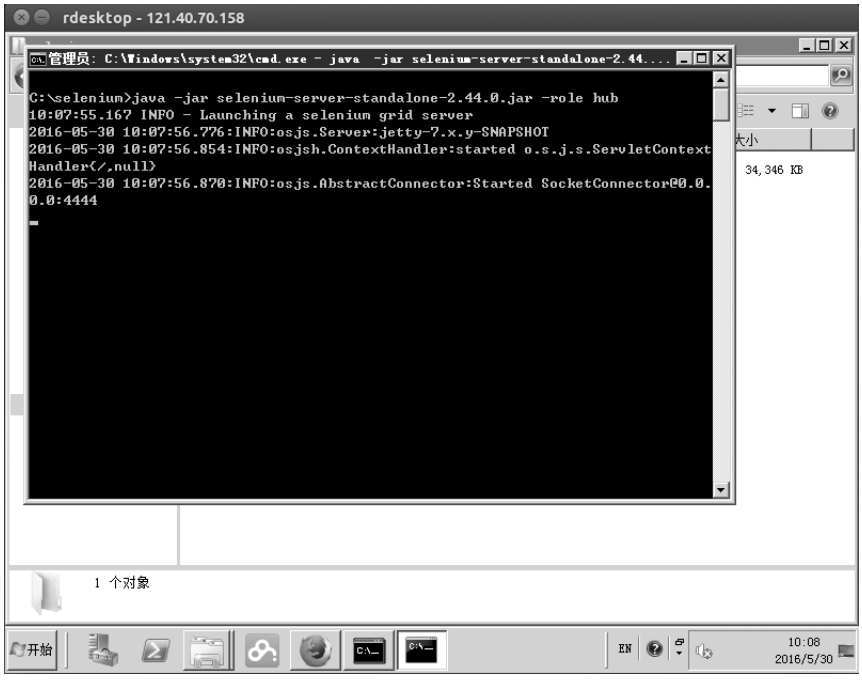


图 5-21　启动 Selenium hub

```
java -jar selenium-server-standalone-2.44.0.jar -role webdriver -hub http://121.40.70.158:4444/grid/register
```

启动后的界面如图 5-22 所示。

注意，图中 cmd 窗口内的信息显示本 node 已经与 hub 连接成功，在最下面的矩形框显示本 node 已经在 hub 上注册成功。

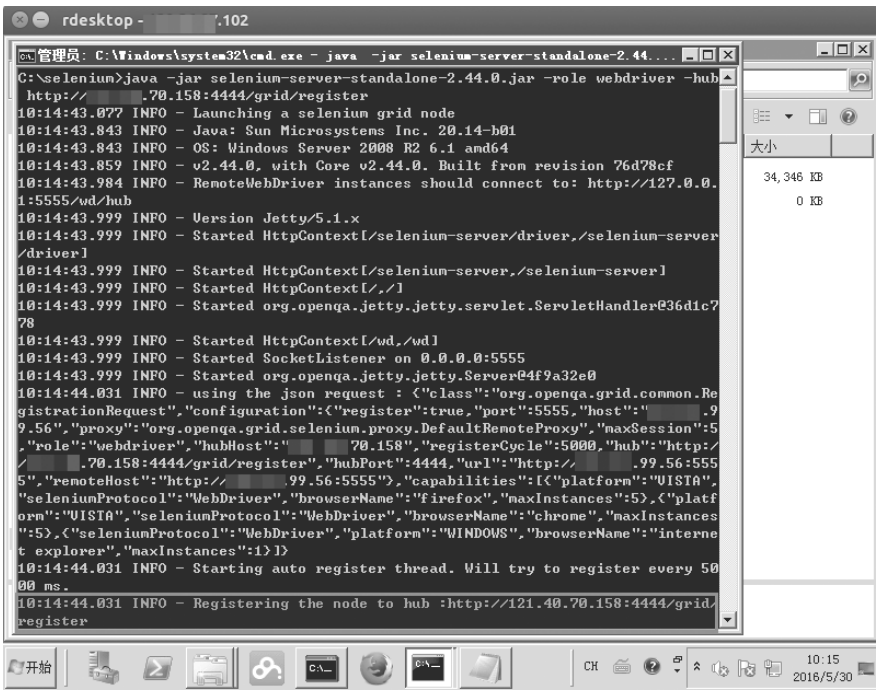


图 5-22　各节点的启动 1

接下来就可以部署其他 3 个 nodes 了，方法与上面相同，如图 5-23、图 5-24 和图 5-25 所示。

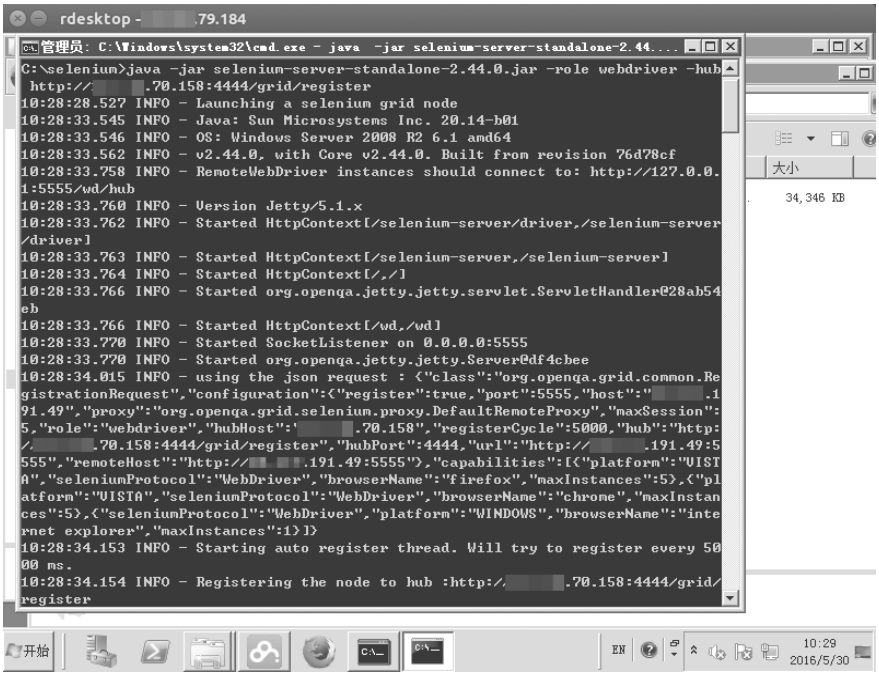


图 5-23　各节点的启动 2

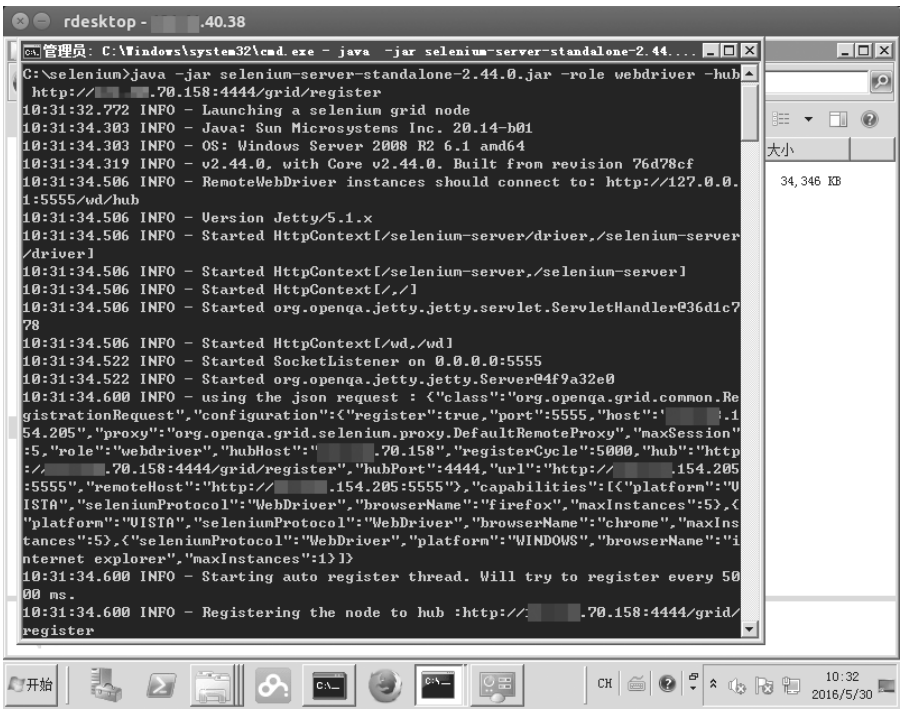


图 5-24　各节点的启动 3

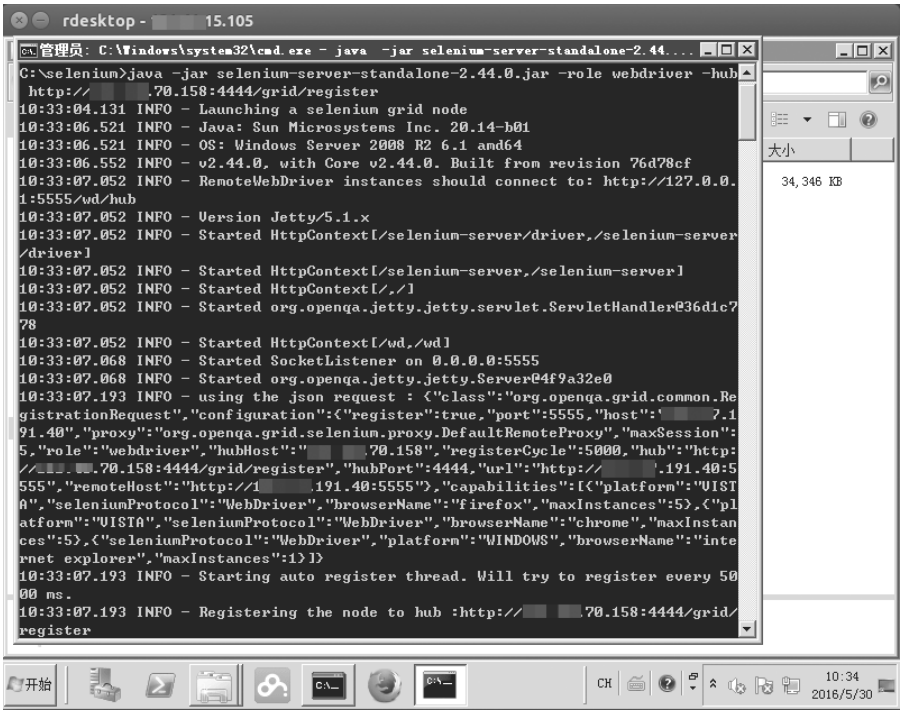


图 5-25　各节点的启动 4

通过上面的配置分布式抓取环境就搭建完毕了。下面的代码是依赖于该环境的抓取代码，仍以前面介绍的多线程抓取实例作为基础。

抓取代码为

```
from selenium import webdriver
import time
from bs4 import BeautifulSoup
import threading

remotes_dict = {
    0: "http://xxx.xxx.xxx.102:5555/wd/hub",
    1: "http:// xxx.xxx.xxx.38:5555/wd/hub",
    2: "http:// xxx.xxx.xxx.105:5555/wd/hub",
    3: "http:// xxx.xxx.xxx.184:5555/wd/hub"
}

def format_str(s):
    return s.replace("\n","").replace(" ","").replace("\t","")

def get_urls_in_pages_by_selenium(ind,from_page_num,to_page_num):
    remote_url = remotes_dict[ind]

    firefox_profile = webdriver.FirefoxProfile()
    firefox_profile.set_preference("permissions.default.stylesheet",2)
    firefox_profile.set_preference("permissions.default.image",2)
    firefox_profile.set_preference("permissions.default.script",2)
    firefox_profile.set_preference("permissions.default.subdocument",2)
    firefox_profile.set_preference("javascript.enabled",False)
    firefox_profile.update_preferences()
    with lock:
        browser = webdriver.Remote(
                    browser_profile = firefox_profile,
                    desired_capabilities =
                                webdriver.DesiredCapabilities.FIREFOX,
                    command_executor = remote_url)

    urls = []
```

```
    search_word = '计算机'
    url_part_1 = 'http://www.phei.com.cn/module/goods/' \
                 'searchkey.jsp?Page='
    url_part_2 = '&Page=2&searchKey='
    for i in range(from_page_num,to_page_num + 1):
        urls.append(url_part_1
                    + str(i) +
                    url_part_2 + search_word)
    all_href_list = []
    for url in urls:
        print(url)
        # resp = requests.get(url)
        # bs = BeautifulSoup(resp.text)
        browser.get(url)
        bs = BeautifulSoup(browser.page_source)
        a_list = bs.find_all('a')
        needed_list = []
        for a in a_list:
            if 'href' in a.attrs:
                href_val = a['href']
                title = a.text
                if 'bookid' in href_val and 'shopcar0.jsp' \
                        not in href_val and title != '':
                    if [title,href_val] not in needed_list:
                        needed_list.append([format_str(title),
                                            format_str(href_val)])
        all_href_list += needed_list
    all_href_file = open(str(from_page_num) + '_' +
                         str(to_page_num) + '_' +
                        'all_hrefs.txt', 'w')
    for href in all_href_list:
        all_href_file.write('\t'.join(href) + '\n')
    all_href_file.close()
    print(from_page_num,to_page_num,len(all_href_list))
    browser.quit()

def multiple_threads_test():
```

```
    t1 = time.time()
    page_ranges_lst = [
        (1,10),
        (11,20),
        (21,30),
        (31,40),
    ]

    th_lst = []
    for ind,page_range in enumerate(page_ranges_lst):
        th = threading.Thread(target = get_urls_in_pages_by_selenium,
                              args = (ind,page_range[0],page_range[1]))
        th_lst.append(th)

    for th in th_lst:
        th.start()

    for th in th_lst:
        th.join()

    t2 = time.time()
    print("使用时间:",t2 - t1)
    return t2 - t1

if __name__ == '__main__':
    lock = threading.Lock()
    multiple_threads_test()
```

首先从 selenium 引入 webdriver，主要用来在后面初始化远程浏览器。remotes_dict 是一个远程地址字典，记录了远程地址。键是一个整数值表示地址的序号，因为在实现中会使用多线程技术，所以需要引入 threading。

format_str()函数主要是用于去除字符串中的回车、空格和 tab 符号。

get_urls_in_pages_by_selenium 函数实现了初始化远程浏览器，并可利用远程浏览器的抓取功能。该函数接受三个参数。其中，ind 表示远程地址的整数序号，from_page_num 和 to_page_num 分别给出了需要抓取页面的起始和终止页号。

```
    remote_url = remotes_dict[ind]
```

根据 ind 得到远程地址。使用多线程实现时，在利用 get_urls_in_pages_by_selenium 函数

初始化线程时传入。remote_url 就是远程地址，后面就是在这个地址上初始化浏览器。接下来的代码用于在初始化 Firefox 浏览器时定制浏览器的功能。

```
firefox_profile = webdriver. FirefoxProfile( )
firefox_profile. set_preference( "permissions. default. stylesheet" ,2)
firefox_profile. set_preference( "permissions. default. image" ,2)
firefox_profile. set_preference( "permissions. default. script" ,2)
firefox_profile. set_preference( "permissions. default. subdocument" ,2)
firefox_profile. set_preference( "javascript. enabled" ,False)
```

"permissions. default. stylesheet"是 Firefox 浏览器的一个属性，当值为 2 时，可以关掉浏览器的样式表功能;"permissions. default. image"也是一个属性，当值为 2 时，可以使浏览器在网页加载时不下载图片，这样就减少了网页的加载时间，同时减少了抓取流量;"permissions. default. script""permissions. default. subdocument"和"javascript. enabled"的设置可以关闭浏览器的脚本执行功能，因为在该抓取实例中只需得到页面上的文本信息，并且这些信息无需动态加载，所以可以将 Firefox 的脚本执行功能关闭，同样可以减少网页的加载时间。在实际使用中，如果在抓取时希望得到动态加载内容，则不能关闭这些功能，否则将无法得到需要的内容。

```
firefox_profile. update_preferences( )
```

该语句使设置生效。下面的工作就是初始化远程浏览器。

```
with lock:
    browser = webdriver. Remote(
                browser_profile = firefox_profile,
                desired_capabilities =
                        webdriver. DesiredCapabilities. FIREFOX,
                command_executor = remote_url)
```

因为我们将使用多线程实现，所以每个线程都会初始化一个远程浏览器。初始化代码前要加上 with lock:，等同于将浏览器的初始化工作置于临界区之中。当初始化后的线程异步并发执行时，如果初始化浏览器不在临界区中进行，就会发生错误。设置临界区会使并发的各线程依次初始化浏览器，从而可避免错误的发生。下面的工作和前面章节的代码类似，可主要实现根据起止页码的数据抓取。注意，在抓取时使用的代码为

```
browser. get( url)
```

这表明可以像在本地一样使用远程浏览器进行抓取，并且得到页面内容的代码也是相同的，即

```
browser. page_source
```

在 multiple_threads_test 函数中是多线程的实现代码，根据 page_ranges_lst 生成线程，每个线程处理一个起始页码的元组，共 4 个元组，初始化线程时使用元组索引和元组中的起止页码值作为参数，可初始化 4 个线程。

运行这段代码，在 Ubuntu 下可以通过双击左侧任务栏中的 rdesktop 图标，这时 4 个 nodes 会在屏幕排开展示，可以观察到 4 个 nodes 的确在并行工作，如图 5-26 所示。

图 5-26　4 个节点的运行情况

同时，在本地可以观察到各 nodes 正确地返回了网页信息，并且程序进行了正确的抽取。

在使用分布式方案抓取时，如果目标网站上的所需内容是由 Javascript 动态生成时，那么可以使用本节提出的方案进行抓取。应该指出的是，hub 不一定需要单独的云端，这里为

了演示方便，突出 hub 和 nodes 的概念，单独使用一个云端建立了 hub，在实际应用中可以将 hub 置于某个 nodes 上，使 hub 和 nodes 共享一个节点。

5.3 Linux 无图形界面使用 Selenium

Selenium 用在抓取时的好处是可以抓取动态内容，完全是按照浏览器的工作方式工作，但有一个问题是工作时需要弹出浏览器，从前面的实例中就可以看到这一点。在 Selenium 分布式抓取的实例中，使用的云端是 Windows 系统，如果云端是 Linux 系统且没有图形界面环境，则由于无法利用 Selenium 启动浏览器，因此无法抓取。本小节将介绍一种可以在无法弹出浏览器的情况下运行 Selenium 的方法，主要利用了 Xvfb。Xvfb 是“X virtual framebuffer”的简称，可以理解为实施了 X11 显示服务协议的显示服务器。与其他显示服务器相比，Xvfb 在内存中可执行所有的图形操作，而不将结果显示在屏幕上。

首先安装 Xvfb。假设使用的系统是 Ubuntu 14.04 桌面系统，安装 Xvfb 后，可以在图形界面环境即弹出浏览器的情况和非图形界面环境即不弹出浏览器的情况下使用 Selenium。安装 Xvfb 使用的命令为

```
sudo apt-get install xvfb
```

安装好，即可启动 Xvfb 服务，命令为

```
sudo Xvfb :123
```

其中，123 代表显示设备，未来使用该设备时，就可以使用这个数字。启动命令如图 5-27 所示。

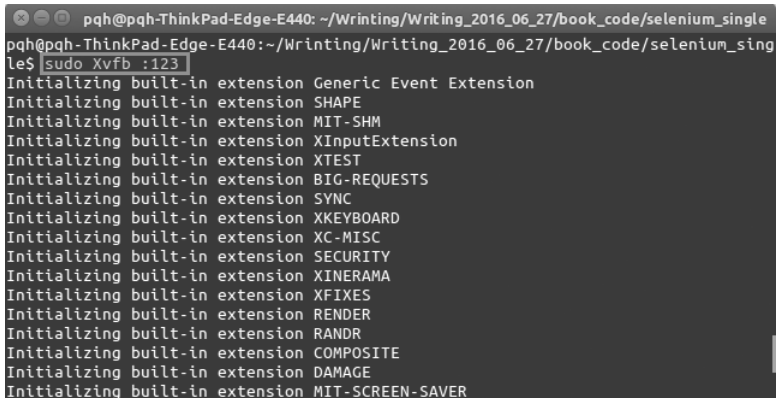


图 5-27　启动命令

下面给出本实例的抓取程序，文件名为 display_test. py。

```
from selenium import webdriver
import time
ff = webdriver. Firefox( )
ff. get( http://www. phei. com. cn/ )
print( len( ff. page_source) ,ff. page_source[0:50] )
time. sleep(5)
ff. close( )
```

该程序使用 Selenium 利用 Firefox 进行抓取，输出 http://www. phei. com. cn/网页的内容长度和前 50 个字符。下面给出正常抓取和使用 Xvfb 服务抓取时的实例。

首先是正常抓取，直接在 shell 中运行 display_test. py 即可。图 5-28 为浏览器界面。图 5-29 为程序运行结果。

图 5-28　浏览器界面

```
pqh@pqh-ThinkPad-Edge-E440: ~/book_code
pqh@pqh-ThinkPad-Edge-E440:~/book_code$ python3 display_test.py
105106 <html xmlns="http://www.w3.org/1999/xhtml"><head>

pqh@pqh-ThinkPad-Edge-E440:~/book_code$
```

图 5-29　程序运行结果

下面使用 Xvfb 服务进行抓取。同样运行上面的程序，但在运行之前，在 shell 中需要设置环境变量

```
export DISPLAY = :123
```

然后启动程序，如图 5-30 所示。

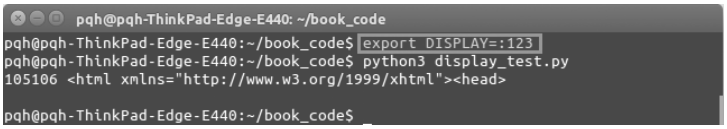


图 5-30　启动程序

在 shell 中输出的结果是一样的，但在抓取的过程中并没有浏览器弹出。注意方框中的环境变量设置。这里只在当前的 shell 中对显示进行设置，如果打开其他 shell 运行上面的程序，则还是会弹出浏览器的。如果希望永久地设置，则保持在抓取时始终不弹出浏览器，可以将 export DISPLAY = :123 写在 shell 的配置文件中。

实战

（1）前面介绍的 Selenium 只支持 get 方法发送请求。selenium - requests 包的网址

```
https://pypi. python. org/pypi/selenium - requests/
```

提供了将 Selenium 和 requests 相结合的类，可以使用 Selenium 发出 post 请求，下载这个包，安装或直接使用 pip3 安装，阅读文档并设计程序，使 Selenium 发出 post 请求。

（2）部署和实现分布式 Selenium 抓取模式。如果没有云端，则可以在局域网环境或者使用虚拟机搭建一个有两个机器的局域网来进行实验。

6 第6章 神秘的 Tor

Tor 是一种可以运行于 Window、Mac、Linux/Unix 和 Android 系统的免费和开源的网络工具，使用 Tor 可以防止他人了解到我们的位置和访问习惯。本章将介绍 Tor 在数据抓取时的作用。无论是单机环境还是分布式环境，都可以通过 Tor 发出请求和得到响应。因为 Tor 可以将我们发出的请求在接入的网络中传送，然后通过在某个也接入到 Tor 网络系统中的机器将请求发送到目标站点，得到响应后，Tor 再将响应传回到我们的机器。这个过程的时间要比常规的访问方式要长，但有一个特点是可以控制 Tor。Tor 就像一个“IP 泵”一样，源源不断地产生各种 IP，为我们发送请求。

本章将介绍使用和控制 Tor 的基本方法。至于 Tor 的使用原理和全部功能和服务，感兴趣的读者可以查找相关的资料。本章的 Tor 就是一个黑盒，将从 IP 控制的角度利用各种方式来使用它，其中有些内容是作者的尝试，读者在使用时可以结合实例观察和思考，找出适合自己的正确使用方式。

6.1 抓取时 IP 被封锁的问题

一般在抓取时，尽量不要给目标网站造成太大的压力。通常，网站会对访问设置一定的规则，对于抓取来说，主要面对的问题是对抓取频次的限制。当抓取过快时，如果超过这一限制，网站就会进行阻止和封锁。这时一般的情况是返回一个验证码输入界面，要求输入验证码继续访问，或者直接返回显示错误信息的 HTML 页面等。阻止和封锁的方法有多种，有的网站会根据 session 的内容来阻止访问，一般都是根据某个 cookie 值来进行判断；有的会对 IP 进行封锁，可能是根据 header 文件中的 X - Forwarded - For[1]，而有些则会封锁真正的 IP 地址。不同的网站有不同的封锁策略，主要表现为在一段时间内，用户无法访问，直至网站解除对此 IP 的封锁。

在遇到 IP 被封锁的时候，可以根据情况采取不同的解决方案。常见的方案如下：

（1）如果采用 ADSL 拨号网络，则可以断开网络后再次连接，这样在动态环境下可能会

分到新的 IP 地址，继续抓取。缺点是断开重连这种方式不能保证下一次分得的 IP 地址与本次 IP 地址是不同的。此外，如果不改变抓取的频次，那么即使下次分得新的 IP 地址，还是会被封锁。一般可以在 ADSL 环境下对代码进行测试，对网站的封锁机制进行测试，调整代码相关的参数，降低被封锁的可能性。

（2）可以在网上搜索一些免费代理，使用这些代理完成抓取。如果可用的代理很多，则可以考虑使用多线程或多进程的并发或并行抓取策略，将任务分解，分配到多个代理上完成。这种方式的问题在于，如果不能获得持续时间久、连接稳定的可靠代理，不但会浪费时间，还会因为并发或并行的策略设计使抓取变得复杂和混乱。

（3）可以租用一些具有独立 IP 地址的机器，如阿里云端。这些云端机器装配的系统包括 Linux 和 Windows。硬件配置根据价位的不同分为很多种。对于抓取来说，主要是考虑网络带宽这一 I/O 因素，而对于其他的配置没有太高的要求。尽管在单核的机器上 Python 的多线程没有优势，但对于抓取这类非计算密集性任务，多线程或多程序的并发执行也可以提高效率。因为有多台机器抓取，抓取任务被分散了，相对于一台机器来说，不但效率提高了，被封锁的可能性也降低了。这种情况问题仍然存在，目标网站的策略是封锁真实的 IP 地址，在一段时期内，如果利用多个云端并行抓取目标网站来完成任务，那么被封锁的云端将失去作用。如果在并行执行任务的过程中，各云端的负载基本均衡，各云端的抓取行为相似，那么封锁 1 个就可能会被封锁全部。在封锁期间，这些云端将不能够被用来继续抓取，将无法完成任务。这也是在第一种方案中提到的测试的重要性，在实际部署于云端前，应先了解目标网站对频次的限制，减小未来被封锁的可能性。

（4）根据 session 来封锁，依据某些 cookie 值或 http 请求头文件中的字段，如 X - Forwarded - For' 的值进行封锁，则可以通过代码的方式来修改相应的请求字段值。封锁的机制是通过实验探测出来的，在被封锁时，可以对相关参数进行修改后继续抓取，检测网站是否根据这些参数进行封锁。如果不是，就可以根据实际情况采取上面提到的策略。如果是，就可以直接使用代码在抓取时设置相应的参数来解决问题。

下面将介绍一种终极的防封锁抓取方案，就是使用 Tor 作为代理实施抓取。

6.2 Tor 的安装与使用

对于 Tor 的安装，可以根据不同的操作系统下载相应的文件，按安装说明一步步安装即可。下面介绍在 Ubuntu 下的使用方法。

在 Ubuntu 下，双击 Tor 的图标，出现如图 6-1 所示的界面。

随着进度条的前进，Tor 会进行一些相关的设置。如果这个过程出现问题，则无法成功

连接并打开浏览器，此时需要打开网络设置，进行一些参数的配置。其方法是重新双击图标，在出现上面的界面后，单击“Open Setting”按钮，将出现如图 6-2 所示的界面。

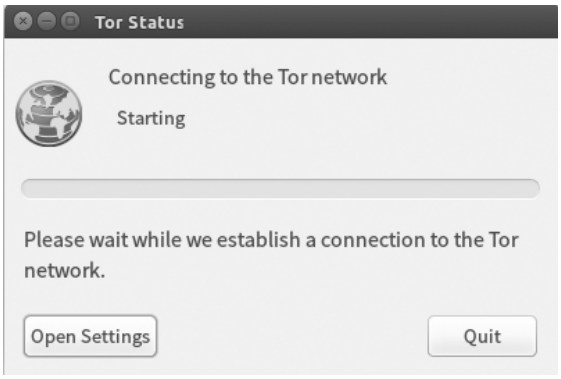


图 6-1　Tor 的连接状态

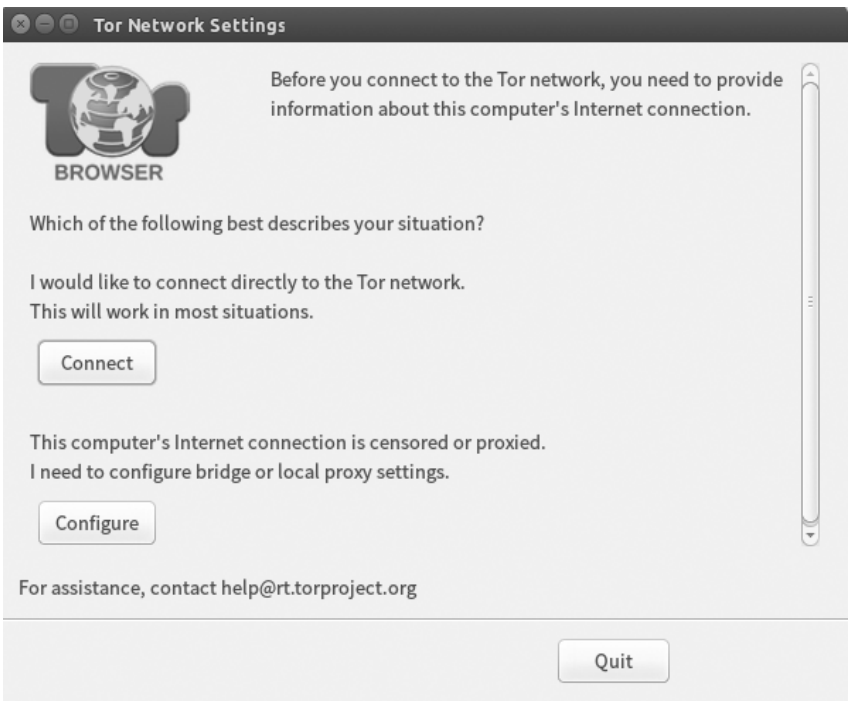


图 6-2　Tor 的网络设置入口

图 6-2 为 Tor 的网络设置入口，因为现在 Tor 还无法成功连接，因此如果单击“Connect”的话，则还会出现前面讲述的情况。此时应单击“Configure”进行参数的配置。单击“Configure”后的界面如图 6-3 所示。

在如图 6-3 所示界面选择“Yes”，单击“Next”，在出现的界面上单击“Transport type”下拉框，此时界面如图 6-4 所示。

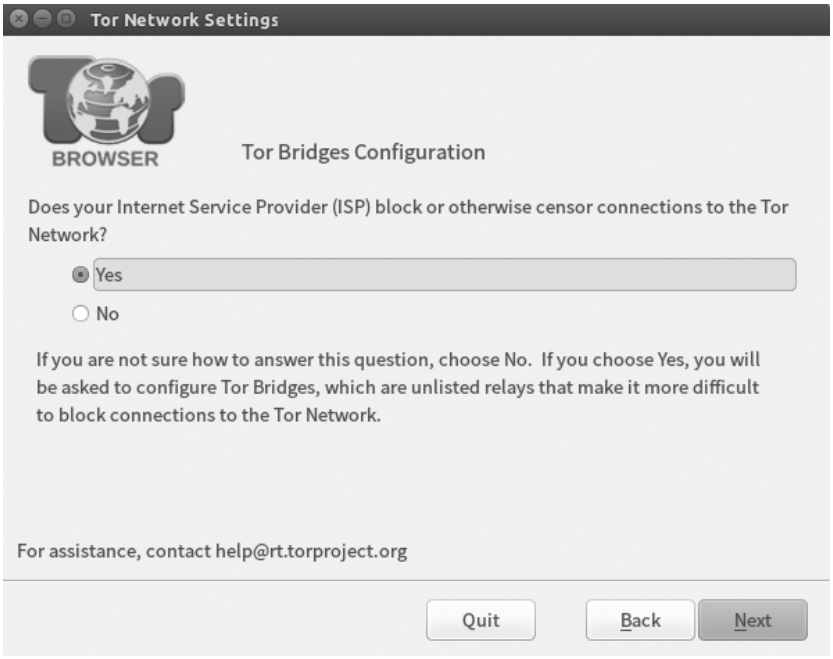


图 6-3　Tor 网络设置选项 1

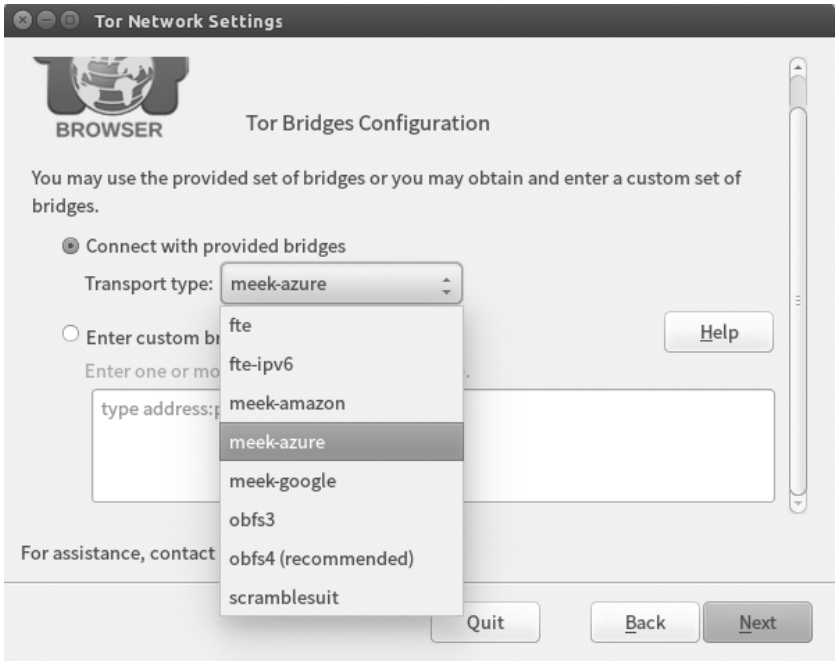


图 6-4　Tor 网络设置选项 2

在如图 6 -4 所示界面中列出了可以选择的传输类型。其中，“meek - azure” 和 “meek - amazon” 是经常可以连通的传输类型，此处选择 “meek - amazon”。继续单击 “Next”，

出现的界面如图 6-5 所示。该界面询问是否计算机在接入互联网时使用了本地代理，如果没有的话，选择“No”。单击“Next”，此时又会出现连接状态图，如果设置正确，则进度条会一直向前，直至启动 Tor 浏览器。启动后的界面如图 6-6 所示。

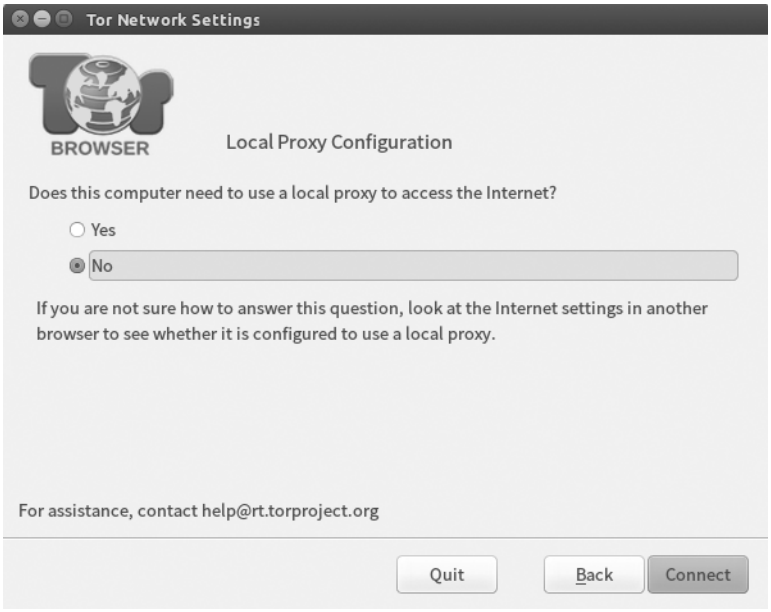


图 6-5　Tor 网络设置选项 3

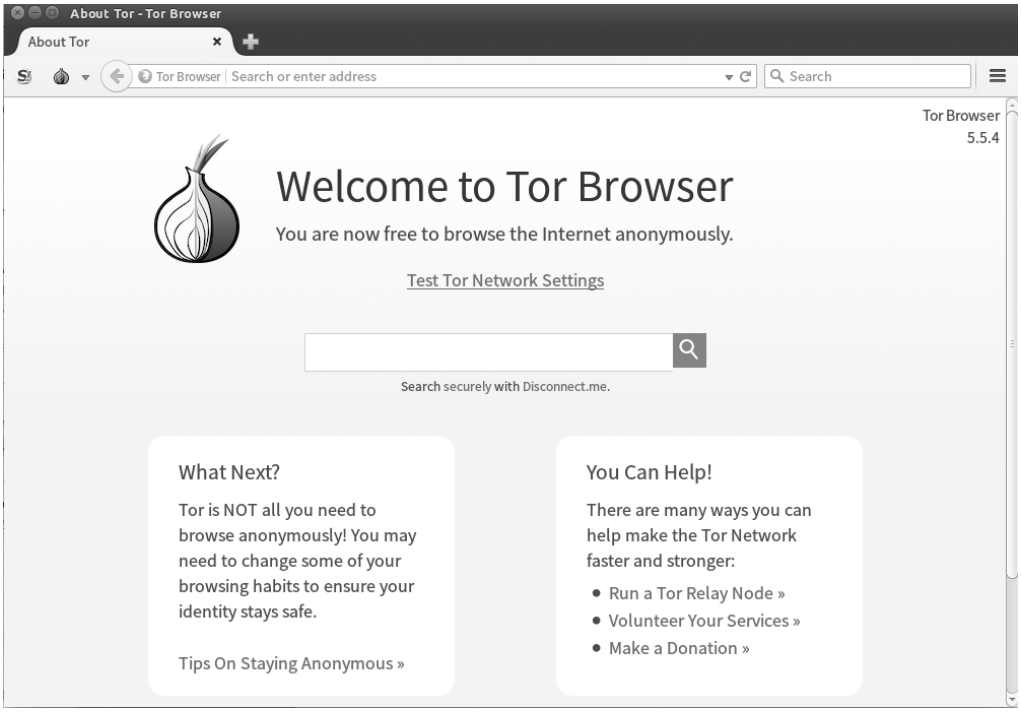


图 6-6　启动 Tor 后的界面

在如图 6-6 所示中的浏览器地址栏内输入“www. google. com”，就可以利用 Tor 访问 google 了。为了查看 Tor 连接到 google 的路径，可以单击地址栏左侧洋葱图标右侧的箭头，单击后的界面如图 6-7 所示。

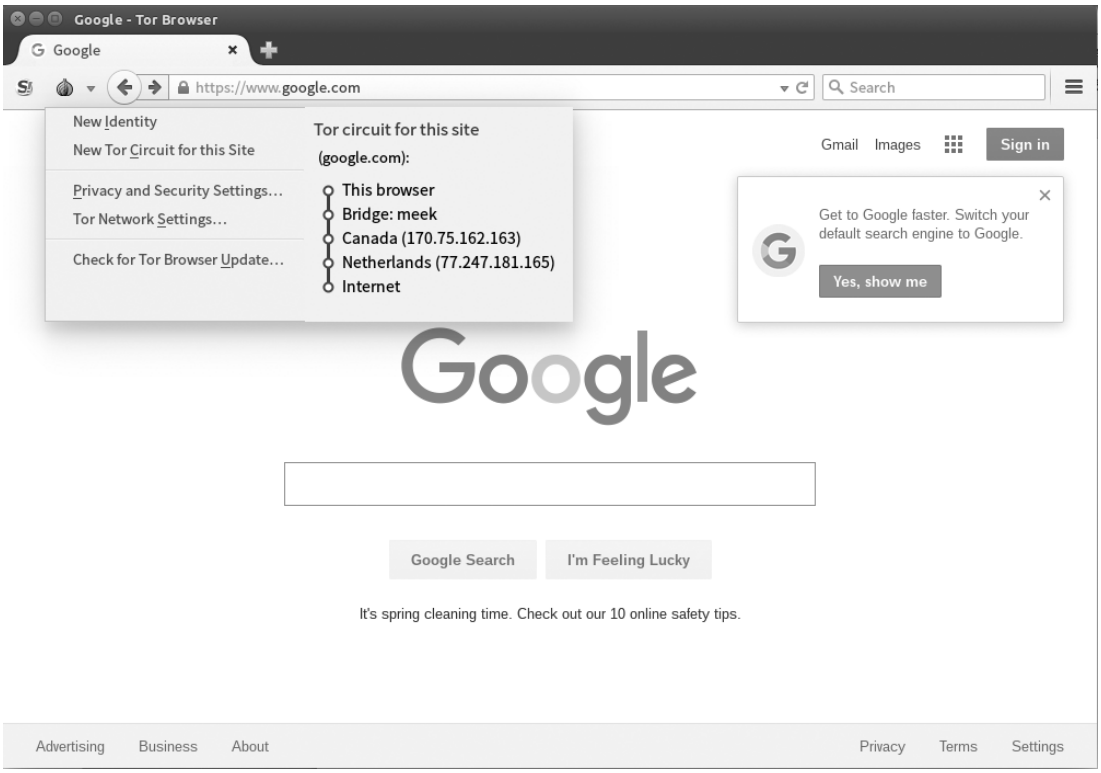


图 6-7　访问 Google 的界面

单击图 6-7 后弹出的放大图片如图 6-8 所示，框内就是为了访问 www. google. com 的 Tor 浏览器 IP 转换路径，在 Windows 下，使用方法与 Ubuntu 相同。

图 6-8　IP 访问路径

6.3　Tor 的多线程使用

使用 Tor 的好处是可以帮助我们在访问网站时使用各种动态的 IP。前面提到过，在访问一些网站时，可能会因访问频次问题造成网站对访问的封锁，如果封锁的方式是阻止真实的机器 IP 地址，那么就难办了。网站一般会对 IP 的封锁时间段进行设置，未来在这段时间内，机器将无法访问该网站。对于这种情况，可以采用 ADSL 线路，断开网络重新连接，获得一个新分配的 IP 地址，但也可能还是上次那个 IP 地址。也可以采用在网络上寻找各种免费代理的方式获得多个免费 IP 地址，但这些免费代理往往连接不可靠，可用时间太短，不断的测试和获取这些免费代理反而加大了抓取的复杂程度。也可以购买一些云端机来搭建分布式抓取网络，在估计好抓取频次参数并进行相应的设置后，可以将任务分解部署到各个云端上，但需要一定的费用。总之，各种传统的方式在解决问题时总是存在着各种问题。本节提到的 Tor 可以解决这个问题，利用 Tor 实施抓取时具有以下特点：

（1）可以生成不同的可用 IP 地址对目标进行访问；

（2）本机操作相对于各种免费代理更加可靠和稳定；

（3）无需费用，Tor 是一种免费的软件。

但 Tor 也有自身的问题，就是对网络环境很敏感，会出现连接不通、连接不可靠、响应缓慢的问题。不通和不可靠主要是由网络环境决定的。目前 Tor 比较稳定了，利用前面使用的 meek - azure 和 meek - amazon 两种方式可以获得比较好的连接效果。响应缓慢是由 Tor 的自身特点决定的，网络上有通过对 Tor 进行参数设置，甚至改变参数值再重新编译的方法，感兴趣的读者可以参考。下面给出 Tor 应用的简单实例，在此基础上给出多线程使用方式。

下面的代码完成的主要工作是在 Tor 的协助下进行数据抓取，抓取的对象是电子工业出版社的首页，网址为“http://www.phei.com.cn/”。抓取使用的是 requests 库，每一次抓取会使用不同的 IP 地址，也就是网站看到的我们每次发出请求的 IP 地址是不同的，requests 自身无法达到更改 IP 地址的效果，需要借助 Tor，利用 Tor 作为访问代理，并利用 stem 库（控制 Tor 的库）中的相关方法，每次抓取后都向 Tor 申请新的 IP 地址。在每一次抓取首页前，先打印出这次抓取使用的 IP 地址，通过访问“http://jsonip.com/”实现。该网站返回的 json 信息中包含本次访问的 IP 地址，如图 6-9 所示。

下面的代码给出了使用 Tor 重复抓取 5 次 http://www.phei.com.cn/的实例。保存文件的命名方法为 IP 地址 +“_pub.html”。每抓取一次就改变一次 IP 地址。在每次抓取的过程中记录下抓取网页并改变 IP 地址的时间。为了使用和控制 Tor，需要安装 socks 和 stem 模块。

图 6-9　访问 http://jsonip.com/的显示结果

```
import socks
import socket
from stem.control import Controller
from stem import Signal
import requests
import time

counter = 0
controller = Controller.from_port(port = 9151)
controller.authenticate()
socks.setdefaultproxy(socks.PROXY_TYPE_SOCKS5,"127.0.0.1",9150)
socket.socket = socks.socksocket

url_ip = 'http://jsonip.com/'
url_pub = 'http://www.phei.com.cn/'
while counter < 5:
    try:
        r = requests.get(url_ip)
        ip_val = r.json()['ip']
        print(counter,'当前 ip:',ip_val)

        time1 = time.time()
        r2 = requests.get(url_pub)
        time2 = time.time()
        print(counter,'抓取时间:',time2 - time1)
        with open(ip_val + '_pub.html','w',encoding = 'utf8') as f:
            f.write(r2.text)

        time3 = time.time()
        controller.signal(Signal.NEWNYM)
        time.sleep(controller.get_newnym_wait())
```

```
            time4 = time. time( )
            print( counter', 改变 ip 时间:' ,time4 - time3)
            counter = counter + 1
        except Exception as e:
            print( e)
```

可以多次运行这个程序，每次抓取和更换 IP 地址的时间都不相同，下面给出两次抓取后的结果。第一次抓取的 5 个首页文件如图 6-10 所示。通过文件大小和缩略图可以看出，requests 已经被成功抓取下来了。第二次抓取的 5 个首页文件如图 6-11 所示。

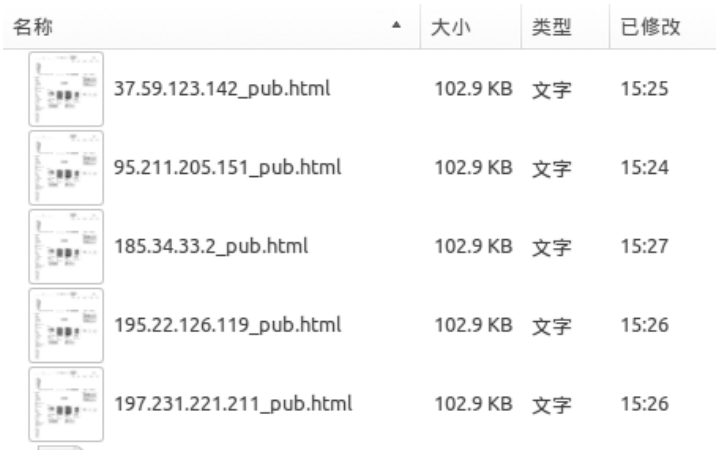


图 6-10　利用 Tor 抓取的 5 个首页文件

0 当前 IP 地址：95. 211. 205. 151
0 抓取时间：13. 138118743896484
0 改变 IP 地址时间：10. 002214193344116
1 当前 IP 地址：37. 59. 123. 142
1 抓取时间：8. 46153450012207
1 改变 IP 地址时间：10. 007575511932373
2 当前 IP 地址：197. 231. 221. 211
2 抓取时间：51. 57068610191345
2 改变 IP 地址时间：10. 002117156982422
3 当前 IP 地址：195. 22. 126. 119
3 抓取时间：16. 691622734069824
3 改变 IP 地址时间：10. 003002166748047
4 当前 IP 地址：185. 34. 33. 2
4 抓取时间：18. 085627794265747

4 改变 IP 地址时间：10.005176305770874

图 6-11　利用 Tor 第二次抓取的 5 个首页文件

0 当前 IP 地址：35.0.127.52

0 抓取时间：16.11019992828369

0 改变 IP 地址时间：10.009408950805664

1 当前 IP 地址：212.21.66.6

1 抓取时间：17.315757751464844

1 改变 IP 地址时间：10.008612155914307

2 当前 IP 地址：94.242.228.107

2 抓取时间：21.74932026863098

2 改变 IP 地址时间：10.00131893157959

3 当前 IP 地址：193.90.12.87

3 抓取时间：17.1242892742157

3 改变 IP 地址时间：10.010332584381104

4 当前 IP 地址：193.90.12.86

4 抓取时间：8.13011884689331

4 改变 IP 地址时间：10.010416746139526

通过这两次运行可以看出，每次运行都达到了使用不同 IP 地址访问网站的目标。每次的抓取时间变化很大，除了本地网络因素外，主要是因为每次抓取时都利用 Tor 作为代理，使用随机分布于全球的一个 Tor 节点访问网站，并且与 Tor 体系自身的结构特点也有很大的关系。最长的一次抓取时间用了 50 多秒，而最短的用了 8 秒左右。50 多秒的抓取发生在第一次运行中 counter = 2 时，IP 地址为 197.231.221.211，在百度 IP 地址查询，该 IP 地址来自利比里亚。8 秒左右的抓取发生在第二次运行中 counter = 4 时，该 IP 地址来自挪威。不

同的环境运行会有不同的结果，可能某次运行抓取的时间很短，也有可能某次运行出现更长的抓取时间，但无论如何，这些时间都远远超过了通常抓取一个网页的时间。为了提高速度，可以采用多线程方法，这是下面将要讨论的内容。这里需要注意，每次改变 IP 地址的时间为 10 秒左右，由 controller 的 get_newnym_wait( ) 方法决定。下面是该方法的代码和说明，感兴趣的读者可以进行进一步的研究。

```
def get_newnym_wait(self):
    """
    Provides the number of seconds until a NEWNYM signal would be respected.
    This can only account for signals sent via this controller.

    .. versionadded:: 1.2.0

    :returns: **float** for the number of seconds until tor would respect
      another NEWNYM signal
    """

    return max(0.0, self._last_newnym + 10 - time.time())
```

通过对 get_newnym_wait 函数的进一步分析可以得知，在初始化时，self._last_newnym = 0.0，因此 get_newnym_wait 的返回值是 10，time.sleep(controller.get_newnym_wait( )) 实际上等待了 10 秒，如上面输出中的

```
…
2 改变 IP 地址时间: 10.00131893157959
3 当前 IP 地址: 193.90.12.87
3 抓取时间: 17.1242892742157
3 改变 IP 地址时间: 10.010332584381104
…
```

改变 IP 地址时间就体现了这种情况。在实际运行环境中，可以根据网络情况等待时间较少的秒数。比如，将 time.sleep(controller.get_newnym_wait( ))注释掉，然后改为 time.sleep(5)，即

```
#time.sleep(controller.get_newnym_wait( ))
time.sleep(5)
```

表示只等待 5 秒即可。

使用多线程的方法可以使多个任务并发执行，减少平均的抓取时间。下面给出一个多

线程的实例。在该例中，有 4 个线程并发抓取，为了简单起见，这 4 个任务完成的工作是相同的，都是重复抓取 http://www.phei.com.cn/首页 5 次，当 4 个线程完成后，将一共抓取 20 次首页内容。为了演示在多线程的情况下使用 Tor，本例中规定，每并发抓取 5 个页面后，就利用 Tor 更改一次 IP 地址，代码为

```
import socks
import socket
fromstem.control import Controller
from stem import Signal
import requests
import time
from threading import Thread,Lock

url_ip = 'http://jsonip.com/'
url_pub = 'http://www.phei.com.cn/'

cnt = 0

controller = Controller.from_port(port = 9151)
controller.authenticate()
socks.setdefaultproxy(socks.PROXY_TYPE_SOCKS5,"127.0.0.1",9150)
socket.socket = socks.socksocket

defcrawl_with_tor(c,thname):

    globalcnt

    counter = 0
    start_time = time.time()
    while counter < c:
        try:
            r = requests.get(url_ip)
            ip_val = r.json()['ip']
            print(thname,cnt,'当前 ip:',ip_val)

            time1 = time.time()
            r2 = requests.get(url_pub)
```

```
                time2 = time.time()
                print(thname,cnt,'抓取时间:',time2 - time1)
                with open(thname + '_' + ip_val + '_pub.html',
                          'w',encoding = 'utf8') as f:
                    f.write(r2.text)

                with lock:
                    cnt += 1
                    if cnt % 5 == 0:
                        time3 = time.time()
                        controller.signal(Signal.NEWNYM)
                        time.sleep(5)
                        r = requests.get(url_ip)
                        ip_val = r.json()['ip']
                        print(thname,cnt,'改变 ip:',ip_val)

                        time4 = time.time()
                        print(thname,cnt,'改变 ip 时间:',time4 - time3)

                    counter +=+ 1
            except Exception as e:
                print(e)
        end_time = time.time()
        with open(thname + '_finished.txt','w',encoding = 'utf8') as f:
            f.write(str(end_time - start_time) + '\n')

if __name__ == '__main__':

    lock = Lock()

    start_all_time = time.time()
    th_lst = []
    for i in range(0,4):
        th = Thread(target = crawl_with_tor,args = (5,str(i)))
        th_lst.append(th)

    forth in th_lst:
```

```
        th.start()
    forth in th_lst:
        th.join()

    end_all_time = time.time()
        print('总耗时:',end_all_time - start_all_time)
```

首先，引入了 Lock，代码为

```
from threading import Thread,Lock
```

在该程序中需要引入 Lock，因为各线程抓取的数量需要加到全局变量 cnt 中。cnt 是一个临界资源，对其进行加操作时需要使用 Lock，可保证多线程互斥地访问临界资源。代码为

```
lock = Lock()
```

实现了 Lock 的初始化，初始化后的对象赋给了 lock，可以使用 with 语法结合 lock 保证对 cnt 的互斥访问，代码段为

```
    with lock:
        cnt += 1
        if cnt % 5 == 0:
            …
```

实现了抓取完成后对 cnt 的加 1 操作。注意，if cnt % 5 == 0：表示当抓取数量为 5 的倍数时进行 IP 地址的变换。

运行这段代码后，输出的最后部分截图如图 6-12 所示。

```
1 10 改变ip: 193.111.136.162
1 10 改变ip时间:  12.633602380752563
0 11 抓取时间:  14.64713430404663
2 12 抓取时间:  16.407325267791748
0 13 当前ip: 193.111.136.162
1 13 当前ip: 193.111.136.162
3 13 当前ip: 193.111.136.162
0 13 抓取时间:  8.424739360809326
1 14 抓取时间:  11.174721240997314
3 14 抓取时间:  10.918918371200562
0 15 当前ip: 197.231.221.211
3 15 改变ip: 197.231.221.211
3 15 改变ip时间:  7.530170917510986
0 16 抓取时间:  4.8576836585998535
3 17 当前ip: 197.231.221.211
1 17 当前ip: 197.231.221.211
1 17 抓取时间:  4.851041793823242
3 17 抓取时间:  5.110082387924194
1 19 当前ip: 197.231.221.211
1 19 抓取时间:  5.814682245254517
1 20 改变ip: 185.129.62.63
1 20 改变ip时间:  7.177131175994873
总耗时: 167.67316055297852
```

图 6-12　输出的最后部分截图

可以看到，当抓取总数 cnt 为 5 的倍数时，IP 地址发生了改变。该例的思想具有普遍性，可以在并发的情况下抓取一定数量后进行 IP 地址的切换，大大减小了被封锁的可能性。抓取时间视网络情况而定，在并发抓取的方式下完成同样的任务总耗时会大大减少。

6.4　Tor 与 Selenium 结合

前面介绍 Tor 的多线程使用时，使用的是 requests 进行抓取，如果抓取的目标内容中有动态加载内容，则可以使用 Selenium 进行抓取。本节将介绍 Tor 和 Selenium 结合进行抓取的方法。

本例中仍然使用 Firefox 作为抓取浏览器。在使用 Selenium 控制浏览器抓取时，如果与 Tor 结合，并希望通过 Tor 作为代理，则需要对浏览器的相关参数进行设置，主要涉及网络协议的类型和 socks 的相关参数。如果参数设置不正确，则 Firefox 是无法通过 Tor 发出请求和得到响应的。下面给出利用 Tor 作为代理的 Firefox 参数设置方法，代码保存在 tor_selenium. py 中。Tor 在本地启动成功后运行这段代码，在抓取时 Firefox 就会将 Tor 作为代理，利用 Tor 来实现发出请求和接受响应。代码为

```
from selenium import webdriver

url_pub = 'http://www.phei.com.cn/'

profile = webdriver.FirefoxProfile()

profile.set_preference("permissions.default.stylesheet",2)
profile.set_preference("permissions.default.image",2)
profile.set_preference("dom.ipc.plugins.enabled.libflashplayer.so",
                       "false")

profile.set_preference('network.proxy.type',1)
profile.set_preference('network.proxy.socks','127.0.0.1')
profile.set_preference('network.proxy.socks_port',9150)
profile.set_preference('network.proxy.socks_version',5)
profile.update_preferences()

browser = webdriver.Firefox(profile)
```

```
browser.get(url_pub)

with open('tor_selenium' + '.html', 'w',encoding = 'utf8') as f:
    f.write(browser.page_source)

browser.quit()
```

其中，

```
profile.set_preference('network.proxy.type',1)
profile.set_preference('network.proxy.socks','127.0.0.1')
profile.set_preference('network.proxy.socks_port',9150)
profile.set_preference('network.proxy.socks_version',5)
```

为比较重要的设置代码，指定了协议类型及 socks 的 IP 地址和 socks 的端口号。因为 Tor 运行在本地，因此 IP 的地址为 127.0.0.1，而 9150 是 Tor 指定的 socks 端口号。另外，socks 的版本号为 5。

另一个需要注意的配置项为

```
profile.set_preference("dom.ipc.plugins.enabled.libflashplayer.so",
                       "false")
```

表示不加载 flash。

下面启动 Tor，与以前一样，只要双击 Tor 浏览器的图标即可。浏览器正常显示就意味着可以使用这段代码连接 Tor，并使用 Tor 作为代理进行数据的收发。在 shell 中运行上面的代码，会弹出 Firefox 进行网页的访问和抓取，访问时的界面如图 6-13 所示。

图 6-13　使用 Tor 访问时的界面

因为是通过 Tor 代理进行的访问，所以速度没有直接访问网址的速度快。

前面介绍了使用 Tor 的多线程抓取方法，在抓取时可以对 IP 地址进行设置，达到使用变化 IP 地址进行抓取的目的。下面介绍的内容与其类似，都是在抓取时动态地改变 IP 地址，但使用的方式有所差异。本例将启动两个不同的进程：一个负责不断地改变 IP 地址；另一个负责抓取。负责抓取的进程可以使用 requests，也可以使用刚刚介绍的 Selenium。下面以 Selenium 为例说明这种方式。对于改变 IP 地址的进程，其代码在前面已经介绍过。这里为了演示 IP 地址确实被更改了，使用 Selenium 抓取的进程访问 http://jsonip.com/，这样从浏览器的窗口中就可以看到 IP 地址变化的情况。首先给出改变 IP 地址的程序，文件名为 tor_change_ip.py，代码为

```
import socks
import socket
from stem.control import Controller
from stem import Signal
import requests
from bs4 import BeautifulSoup
import time

controller = Controller.from_port(port = 9151)
controller.authenticate()
socks.setdefaultproxy(socks.PROXY_TYPE_SOCKS5,"127.0.0.1",9150)
socket.socket = socks.socksocket

counter = 0
url_ip = 'http://jsonip.com/'
while counter < 500000:
    try:
        r = requests.get(url_ip)
        soup = BeautifulSoup(r.text)
        ip_val = r.json()['ip']
        print(counter,'当前 ip:',ip_val);

        time1 = time.time()
        controller.signal(Signal.NEWNYM)
        # time.sleep(controller.get_newnym_wait())
        time.sleep(5)
        time2 = time.time()
```

```
            print(counter,'改变 ip 时间:',time2 - time1)
            counter += 1
        except Exception as e:
            print(e)
```

代码基本与前面介绍过的相同，为了显示 IP 地址的变化，仍然使用了 requests 向 http://jsonip.com/发出请求，并显示得到的 IP 地址。其中的一个变化是 time.sleep(controller.get_newnym_wait())改为 time.sleep(5)，这样可减少等待的时间。

下面是抓取程序，仍是将 Tor 作为代理，使用 Selenium 进行抓取，抓取的也是 http://jsonip.com/，直接在浏览器中就能显示 IP 地址信息。代码如下，文件名为 tor_selenium_for_chang_ip.py。

```
from selenium import webdriver
import time

url_ip = 'http://jsonip.com/'

profile = webdriver.FirefoxProfile()
profile.set_preference('network.proxy.type',1)
profile.set_preference('network.proxy.socks','127.0.0.1')
profile.set_preference('network.proxy.socks_port',9150)
profile.set_preference('network.proxy.socks_version',5)
profile.update_preferences()

browser = webdriver.Firefox(profile)

while True:
    time.sleep(5)
    browser.get(url_ip)

browser.quit()
```

代码的主要功能就是通过 Firefox 每 5 秒访问一次网站 http://jsonip.com/，无限循环下去。这里的等待时间设置为 5 秒，主要是为了与改变 IP 地址的程序等待时间相对应，实际运行时也可以设置为其他值，而且因为未来这两段代码将各自以线程的形式独立运行，感兴趣的读者可以考虑通过 Python 自身的线程模块或者其他机制，将两个线程设置为以同步的方式运行，即当 IP 地址改变线程将 IP 地址改变后，再运行抓取程序，抓取结束后，再运

行 IP 地址改变程序。这里只是简单地设为两个线程独立运行，没有考虑同步的情况。

下面给出运行的情况。首先启动 Tor，接着分别运行 IP 地址改变程序 tor_change_ip. py 和抓取程序 tor_selenium_for_chang_ip. py，可以分别启动两个 shell 运行两段程序。运行时的截图如图 6-14 所示。

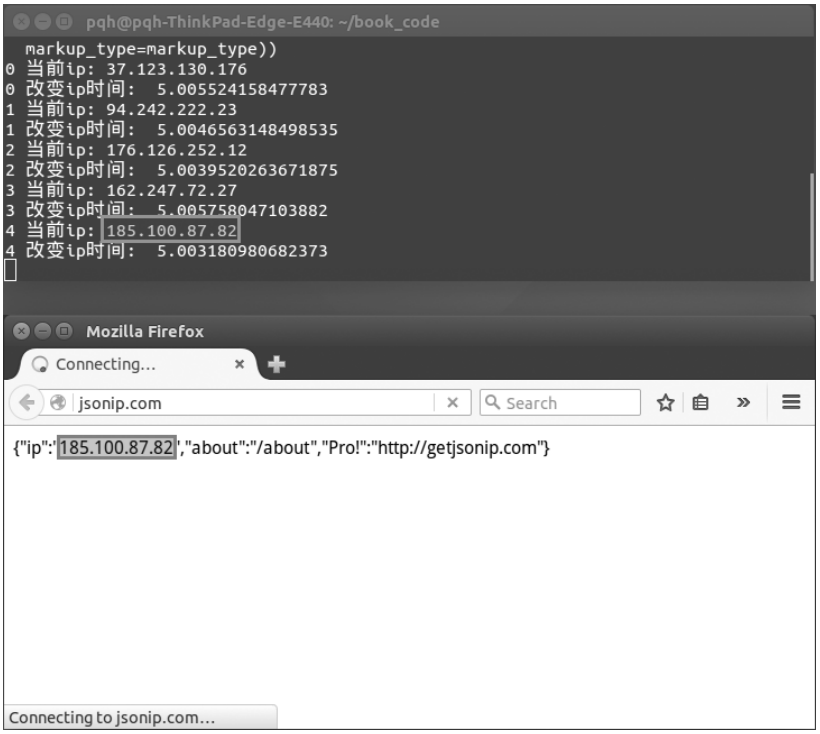


图 6-14　tor_change_ip. py 和 tor_selenium_for_chang_ip. py 运行时的截图

图中可以看到，上面 shell 界面运行的为 IP 地址改变线程，下面运行的是另一个使用 Selenium 进行抓取的程序界面。在此时刻，Selenium 使用的 Firefox 浏览器显示的 IP 地址为 IP 地址改变线程改变的 IP 地址，随着运行就会发现，Firefox 的 IP 地址会不断变化，因为这两个线程不是同步的，所以随着时间的推移，可能出现 Firefox 没有遍历 IP 地址改变线程中所有改变过的 IP 地址，也有可能 Firefox 两次请求显示的是同一地址。正如前面所说，读者可以引入同步机制来解决这个问题。尽管如此，从现在运行的结果来看，IP 地址改变程序就像一个泵一样，源源不断地泵出新的 IP 地址供抓取程序使用。抓取程序总是拥有新的 IP 地址，极大地减少了被封锁的可能。Tor 在本机环境就能为我们提供近乎无穷的 IP 地址。

实战

在本机搭建 Tor 环境，并通过控制 Tor 来获取不同的 IP 地址完成抓取。

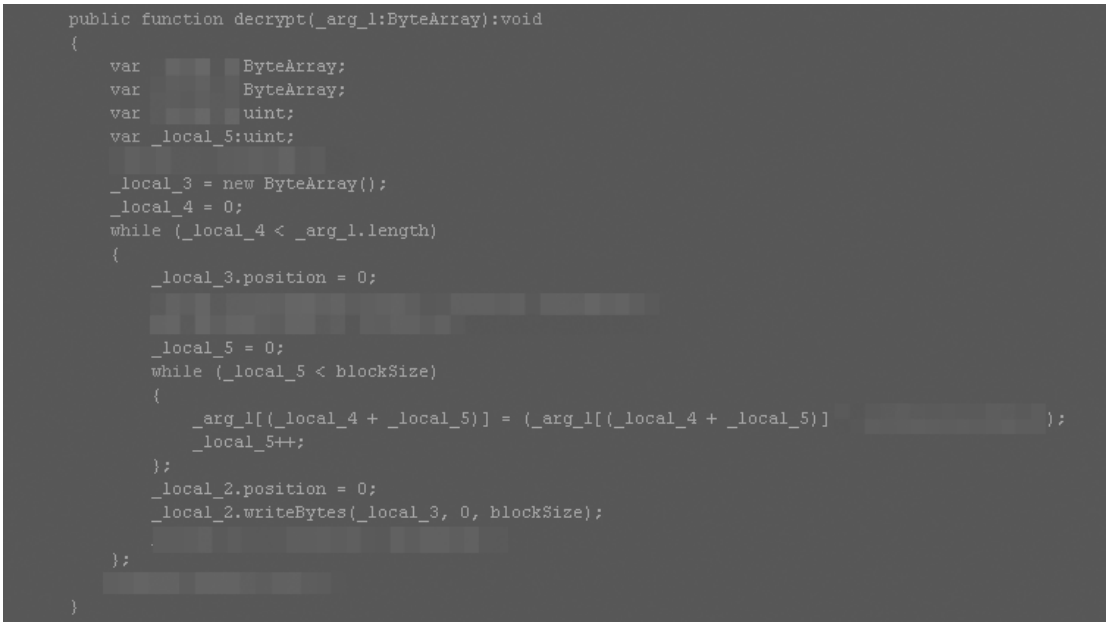
7 第7章 抓取常见问题

本章将考虑一些抓取中常见的问题。这些问题都是抓取时经常会遇到的。

7.1 Flash

很多网站为了保护数据，将数据以 Flash 的方式呈现和展示。在页面加载完成后，Flash 控件会与服务器进行通信，请求需要的数据并将其显示出来。Flash 与服务器之间的通信采用 AMF 协议，与在抓取时对 HTTP 通信协议分析的方式类似，对 AMF 也可使用相似的办法分析出数据请求和响应的格式，然后使用代码模拟这个过程，根据要求自动化整个过程。有时在分析 Flash 与服务器间的通信时会发现，发出的请求已经被加密，同时返回的数据信息也已经被加密，以密文的形式返回。如果不知道秘钥，那么这些信息将无法解密和还原。尽管返回的是密文，但在页面的 Flash 中还是将数据还原和展示给我们。原因在于在页面加载的 Flash 控件中含有加密和解密的代码，可以将自己的请求加密发送出去，将返回的加密信息解密展示出来。因此，可以通过对 Flash 控件的分析找出加密和解密对应的模块，进而分析出秘钥，在拥有秘钥的情况下，整个过程就变得透明和简单了。可以使用 Flash 反编译程序将 Flash 控件反编译成 AS3 代码，然后进行分析。在分析时注意在 Flash 加密时一般都使用 AS3 Crypto 库，该库的网址为 http://crypto.hurlant.com/，里面包含了很多常用的加密和解密算法，在解密的过程中搜索该库的名称，往往可以在反编译后的代码中快速定位到加密和解密的文件位置。图 7-1 为 Flash 反编译后代码。

因为经典加密和解密的算法实施都是类似的，通常在分析出密钥后，就可以使用Python相关的加密和解密库来进行处理。需要注意的问题是，在利用反汇编得到的代码中，尽管可以找到密钥，但要注意观察，利用密钥进行加密或解密位置的前后还会有一些辅助性质的代码，使用 Python 代码还原加密解密过程时也要加上这些代码，而且注意语言间实现的差异。如果涉及 AS3 的补码操作，那么 Python 的补码处理和 AS3 的处理是不同的，涉及此类操作时，需要编写一些 Python 的辅助代码来解决。



```
public function decrypt(_arg_1:ByteArray):void
{
    var      ByteArray;
    var      ByteArray;
    var      uint;
    var _local_5:uint;

    _local_3 = new ByteArray();
    _local_4 = 0;
    while (_local_4 < _arg_1.length)
    {
        _local_3.position = 0;

        _local_5 = 0;
        while (_local_5 < blockSize)
        {
            _arg_1[(_local_4 + _local_5)] = (_arg_1[(_local_4 + _local_5)]                    );
            _local_5++;
        };
        _local_2.position = 0;
        _local_2.writeBytes(_local_3, 0, blockSize);

    };

}
```

图 7-1　Flash 反编译后代码

7.2　桌面程序

有些网站的用户接口不是以 Web 形式提供给客户，而是通过桌面应用程序展示。这种模式通常称为 C/S 结构。这在分析上带来了一定的问题。通常的分析方式是使用浏览器自带的调试工具进行，在网页与服务器的交互过程中可以追踪到相关的请求和响应信息，基于此进行分析，然后使用代码进行模拟。当用户接口是桌面客户端时，用户程序和服务器之间的交互也可能使用 HTTP 协议，但无法被浏览器捕捉。此外，很多此类以桌面应用程序作为用户接口的数据服务模式都是以授权的形式安装的。同样的应用程序安装在不同的机器上可能就无法启动运行了，主要是因为在需要用户名和密码作为基本的认证手段之外，服务器还要验证与具体机器相关的一些信息，如机器的 MAC 地址、关于硬盘的一些序号信息等，为抓取带来了另一种困难。因为可能应用程序运行在别人的机器上，所以很难创造实验环境，并且别人未必允许我们长时间占用机器调试。总之，面对这种情况，抓取的分析和操作都会比传统的 B/S 结构，即浏览器/服务器模式的抓取困难很多。

下面介绍一种解决这类问题的方式。如果在与服务器的交互过程中 HTTP 协议是以明文方式发送内容的，那么可以使用 Wireshark 进行辅助分析。Wireshark 是一款协议分析软件，可以监控整个系统通过网络和外界进行交互时的协议使用情况和协议的内容。这

款软件的功能十分强大，需要在使用中不断摸索抓取用到的一些基本功能。使用这个软件可以在客户端是桌面应用的情况下，运用类似浏览器调试工具进行请求和响应分析，进而编写出代码进行自动化抓取，当然，也可以分析以浏览器作为接口进行的数据请求和响应。

登录 Wireshark 的官方网站 https://www. wireshark. org/下载如图 7-2 所示。

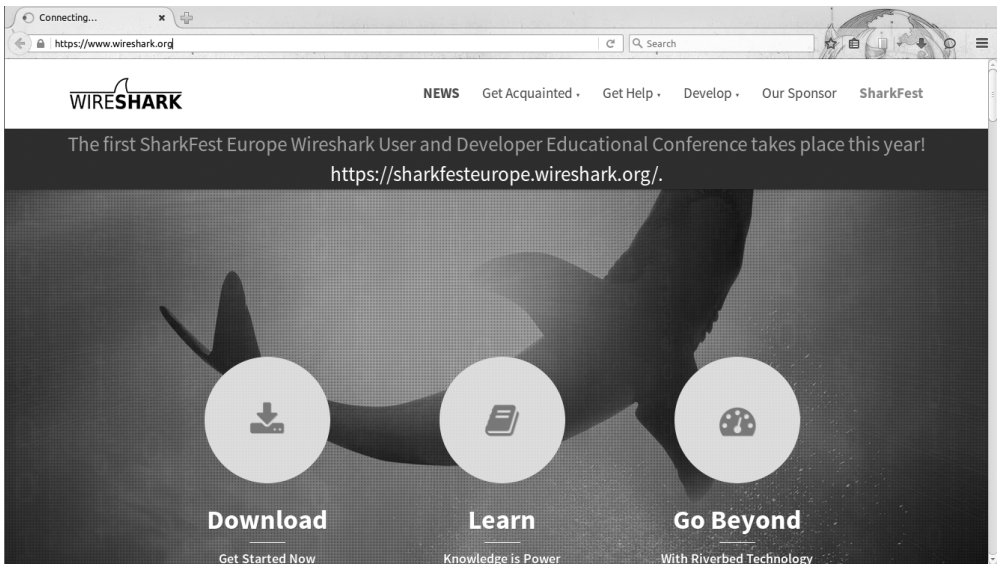


图 7-2　Wireshark 首页

可以根据桌面程序安装的操作系统选择相应的版本，如果运行在 Windows 环境下，就可以选择 Windows 版本的安装程序文件。

可以使用参考教程启动 Wireshark 并进行监控。

图 7-3 给出了使用时协议在不同 IP 地址之间的收发情况。图 7-4 给出了某一条记录里面的协议详细信息。

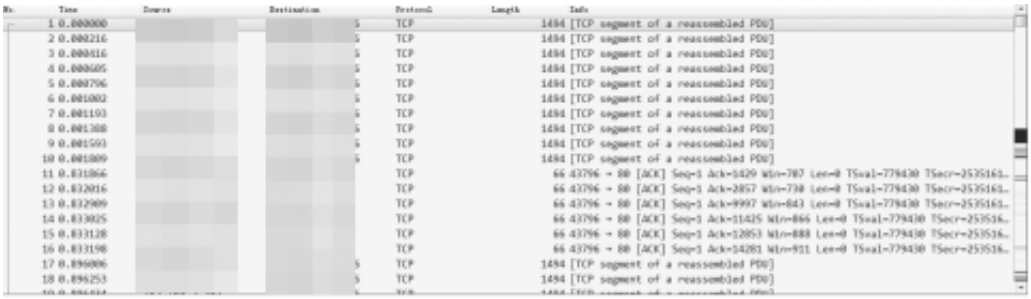


图 7-3　使用时协议在不同 IP 地址之间的收发情况

如果采用逐条协议，以查看数据内容的方式进行分析，那么可能会陷入细节的海洋。

图 7-4　协议详细信息

由于客户端程序是一个黑盒，正如前面提到的，除了基本的用户名和密码的提交以外，可能还涉及授权机器的部分硬件细节，需要什么信息、需要多少信息我们也不确定，如果把精力放到这方面内容的分析和用代码的模拟上，可能会大大提高难度。这里需要注意的是，我们的目的是实现自动化抓取数据，所以应该把焦点放到所关心数据的请求和响应是如何工作的。假如客户端和服务器端是使用 HTTP 协议进行通信的，数据的请求将还是以 get 或 post 的方式发出，那么一种简便的方法是直接在 Wireshark 的协议分析界面，逐步定位到返回数据相关的 get 或 post 方法，然后观察分析，找出得到数据所需要的请求参数。这时的参数就应该是我们在使用代码模拟时所需要的参数了，是经过前面与服务器若干次的交互验证所形成的，是服务器认可的，使用这些参数并做适当的改动，就可以用来与服务器进行交互了。

一般的桌面应用程序客户端都安装在本地，可以进入相应的文件夹进行查看，为了获得对程序的进一步理解，可以对其中的各类库使用工具进行反编译，然后进行导出和分析，因为有时数据的请求和响应也会被加密，与 Flash 控件的反编译一样，可以使用这种方式寻找所需的信息。

7.3　U 盘

一些网站在为用户开通账号或测试账号时，有时会使用一个 U 盘作为辅助登录工具。如果使用 Web 方式对网站进行访问，那么在抓取时 U 盘起的作用并不大。插上 U 盘后，按照常规的方式进行分析，会发现在发送请求的过程中会使用一些参数，这些参数就是通过 U 盘产生的，得到这些参数后就可以使用代码进行抓取了。

7.4 二级三级页面

通常在抓取时遇到的网站形式都是以三级层次组织的：第一级是首页；第二级是列表页；第三级是详细信息页。在对这类网站进行抓取时，很多网站只对二级页面的抓取进行频次的限制，而对三级页面的抓取频次没有限制。在处理这类问题时，可以先将第二级列表页的内容以较为缓慢的频次采集下来。

7.5 图片的处理

有时在抓取时，目标网站可能是出于信息保护的目的，将一些关键的信息，尤其是数字的相关信息以图片的方式展示出来。遇到这种情况时，如果数字在后续的使用中是用来进行计算和统计的，且数字数量较大，则很难用人工的方式将数字一个一个辨出并记下，必须将图片中的数字转为文本数字的形式。转化的方式可利用 PIL 库来执行，或者寻找其他的库，如大多数机器学习库中包含的一些监督式学习算法，经适当修改后也可以用作数字的识别与处理。如果以图片展示数字相关信息不是用来计算的，如电话或手机号码、邮编、QQ 号等，那么可以不必转换，特别是电话或手机号码，此时可以直接将信息以图片形式保存，而且为了更为方便地使用此类信息，可以将图片与其他文字信息混排在一起。图 7-5显示了这样的实例。

在图 7-5 中，第三列的信息是电话号码，全是图片，其余三列是文字信息，Excel 允许这样的存储。Python 有 Excel 的读/写模块，利用这些模块可以方便地对 Excel 信息进行读/写。

7.6 App 数据抓取

通常有两种方式可以解决 App 的抓取问题。第一种方式就是获得 App 应用与服务器交互的 url，然后在机器上利用得到的 url 发出请求，对响应进行抓取即可。另一种方式其实也是通过获得与服务器交互使用的 url，但在发送时同时发出了一些关键的 cookie 信息，在 App 安装的移动设备上可能不便于安装 Web 调试工具，解决的方式是设置 Wi - Fi 热点，使移动设备通过该热点连接网络，利用 Wireshark 对热点进行网络流量的分析，从而抓取 App 通过热点与服务器交互的请求和响应数据。这个步骤与 7.2 节介绍的内容是一致的。下面

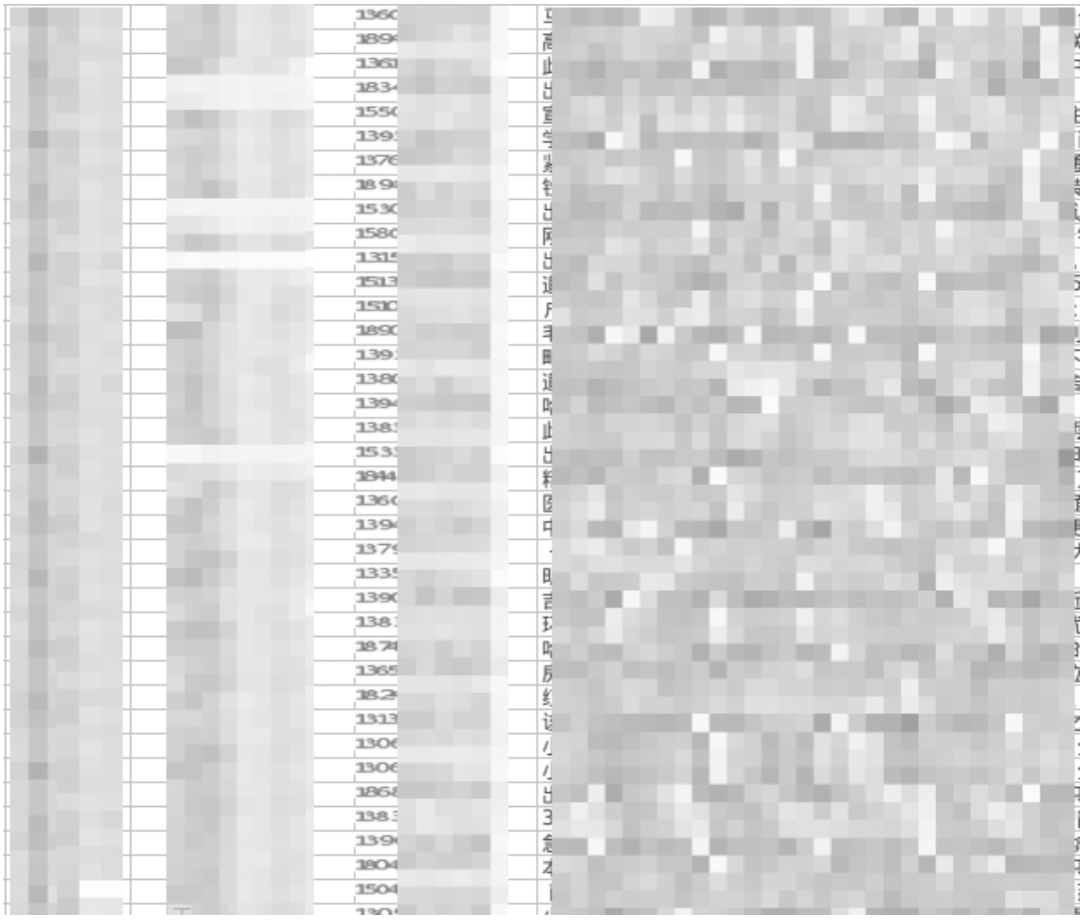


图 7-5　电话号码图片嵌入 Excel

给出在 Windows 环境 Wi - Fi 热点下，使用移动设备连接 Wi - Fi 热点访问互联网，同时用 Wireshark 分析访问的请求和响应的实例。

首先设置 Wi - Fi 热点，操作系统为 Windows 7。

（1）热点设置命令

以管理员身份运行 cmd，打开后，在命令行提示符下分别输入两条命令

```
netsh wlan set hostednetwork mode = allow ssid = myHot key = myHot123
netsh wlan start hostednetwork
```

便可以启动名为“myHot”，密码为“myHot123”的 Wi - Fi 热点，可以使用命令

```
netsh wlan show hostednetwork
```

查看启动结果，如图 7-6 所示。

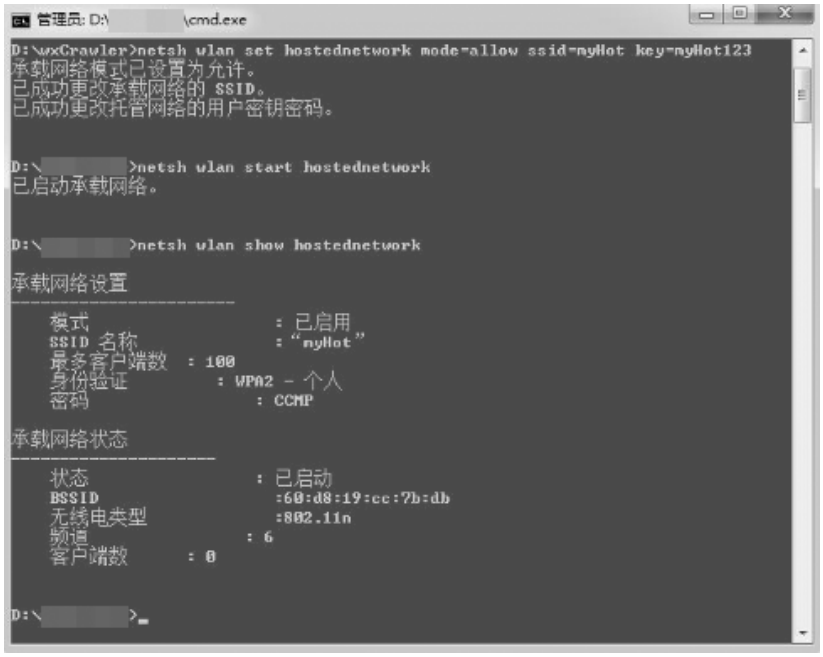


图 7-6　启动结果

（2）查看启动热点

在“控制面板 -> 网络和 Internet -> 网络和共享中心”界面单击左侧的“更改适配器设置”，如图 7-7 所示。

图 7-7　网络共享中心

单击后的界面如图 7-8 所示。注意矩形框中的“无线网络连接 2”，下一步的设置中会使用到这个名称。

图 7-8　虚拟 Wi－Fi 名称

（3）共享设置

为了能够连接到热点，还需要进行现有连接的共享设置。右键单击现有连接的名称，在弹出的窗口中单击“状态”按钮，如图 7-9 所示。在弹出的“无线网络连接 状态”窗口中单击“属性”按钮，如图 7-10 所示。在弹出的属性窗口中选择“共享”标签窗口，勾选第一个复选框，并选择“无线网络连接 2”，单击“确定”按钮，如图 7-11 所示。

图 7-9　单击现有连接的“状态”按钮

图 7-10　单击“属性”按钮

此时，虚拟出的 Wi－Fi 热点 myHot 就可以使用了。

（4）使用移动设备连接热点

本例中使用 ipad mini2 作为移动设备，可以像连接任何 Wi－Fi 设备一样的步骤来连接热点，如果前几步设置正确的话，就可以正确地连接了。

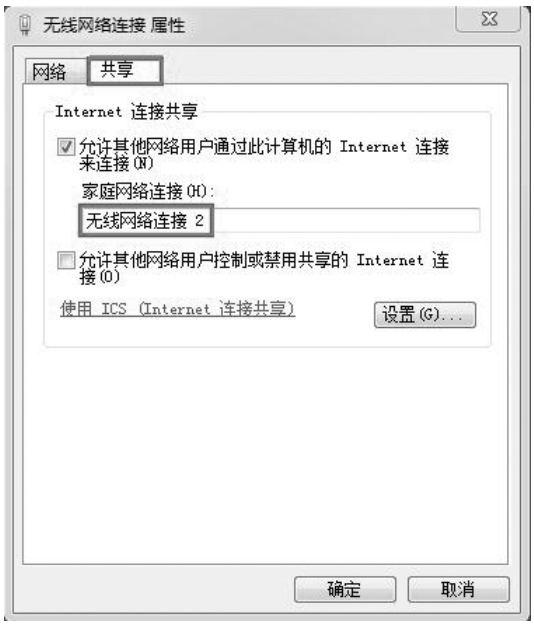


图 7-11　设置共享

（5）启动 Wireshark

在启动的界面上双击“无线网络连接 2”，如图 7-12 所示，启动对该连接的监控。

图 7-12　监控“无线网络连接 2”

（6）监控移动设备的数据

在移动设备上访问“www. phei. com. cn”，此时在上一步双击“无线网络连接 2”所弹出的窗口中，可以看到以下的监控内容，如图 7-13 所示。

在图 7-13 中，在列表项中的方框内显示了“GET/HTTP/1. 1”，这就是向“www. phei. com. cn”发出请求的 HTTP 协议，单击这一列表项，弹出该协议的详细内容，如“Host：www. phei. com. cn\r\n”是服务器，在“User-Agent”中的“AppWebKit”表示了 Safari 浏览器。

随着与服务器之间的交互，Wireshark 会截获更多的内容，会有更多的列表项出现。可以单击列表头的“Protocol”域来对这些列表项按照协议的类型排序，如图 7-14 所示。

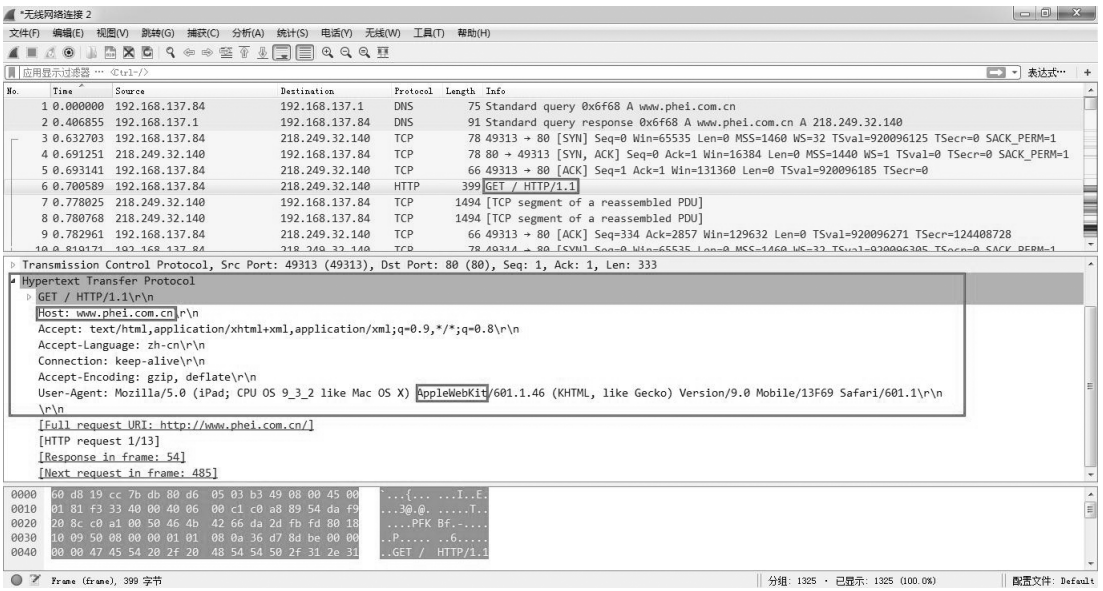


图 7-13　请求的详细信息

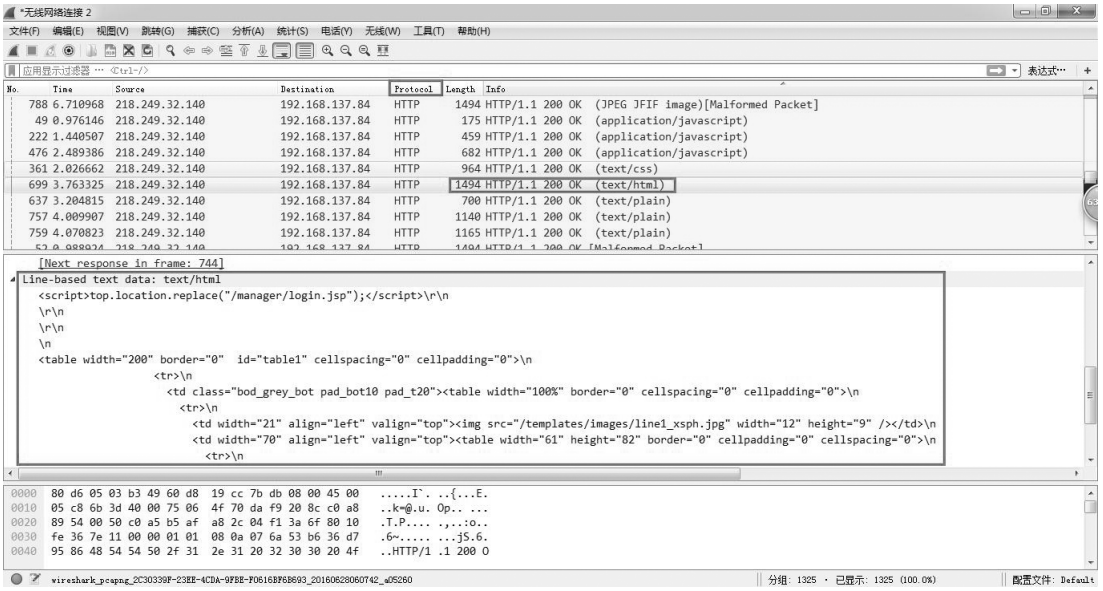


图 7-14　响应的详细信息

进一步观察，可以在“Info”域中观察到“(text/html)”类型的信息，该项的“Length”域为 1494，源网站是“218.249.32.140”，说明这一项的内容是响应内容，内容的长度是 1494 个字节。单击该列表项，会在下面的窗口中显示内容。其中，“Line-based text data：text/html”给出了返回的内容。

这样就可以通过 Wireshark 监控热点，达到对连接热点移动设备的信息监控。本例抓取

了在移动设备上使用浏览器访问网站时的交互数据，如果是 App 应用发送的数据，则也可以使用类似的方法分析。

实战

在 Windows 7 平台上设置 Wi – Fi 热点，使用移动端设备连接该热点，启动 Wireshark，启动移动端设备的网络应用，在 Wireshark 中观察数据的请求和响应过程。

8 第 8 章 监控框架

本章介绍监控框架。在数据抓取过程中，如果能对抓取的数量和抓取的内容质量进行实时的观测，及时地发现和解决出现的问题，则可以极大地提高抓取效率。尽管可以利用记录日志的方式来达到这个目的，但图形化界面的监控系统将会更加直接、更为直观地显示抓取情况，并可提供接口来进一步分析抓取数据的质量。本章将介绍如何搭建一个图形化的数据抓取实时监控框架。该框架侧重于抓取数量的监控。对于抓取数据质量的分析可参考“拥抱大数据”一章中介绍的方法。

对于小规模的抓取一般可以比较快速地完成，此时无需监控系统，抓取结束后，可以通过查看抓取的结果来判断是否出现了问题。比如，是否出现了漏抓；是否已经被网站封锁，但还是在继续抓取，导致抓回了很多无用的页面；是否某个字段的信息是由 Javascript 动态获得的而没有抓取等。对于这些问题可以通过修改程序代码重新运行即可，并不需要搭建监控系统。

如果抓取量比较大且持续时间长，就有必要搭建一个监控系统了。比如，对于周期性持续抓取系统，因为目标网站内容的更新，会按照周期不断地进行抓取。运行这类系统后，就需要一个监控系统对其活动进行监控，利用监控系统可以根据抓取的数据量了解在某段时间抓取工作是否正常，目前系统是否工作正常，进而发现和改正问题。另外，在实时周期性的大数据抓取时，往往采用的是分布式抓取环境。在分布式环境中，由于抓取节点可能位于互联网环境中，因此及时了解各节点的工作情况就十分关键了。比如，可能有的抓取节点意外关机，失去了响应，有的可能负载过大，那么在分配任务时就要考率将任务分配给负载较低的节点。总之，及时地发现和处理出现的问题和故障的节点，是一个在分布式抓取中应该重视的问题。对于这类问题，最好搭建一个实时的监控系统进行监控，以及时地发现并处理问题。

监控的内容包括抓取数据数量和抓取数据质量。

（1）抓取数据数量

抓取数据数量主要包含三个量：目标抓取量、已抓取数据量、有效抓取量。目标抓取量是在抓取时估计的抓取完成量。对于周期性抓取来讲，这个量是不确定的。因为目标网

站的内容不断地更新，抓取时无法估计要抓取的数据总量。后两个量是我们所关注的，是在监控时反映抓取情况的主要指标。对于已抓取数据量来说，一次抓取主要指发出抓取请求并得到响应的过程，这样的抓取数量就是已抓取的数量。有效抓取量是在已抓取数据量中，有效数据的总数量。从不同的角度，有效抓取量有着不同的表现形式。比如，已抓取数据量为 10，在第 6 次抓取后，网站进行了封锁，此后的 4 次抓取虽然能够成功，但抓取到的可能是提示输入验证码的页面，或者是一个类似“访问太过频繁，请稍后再试”的页面，这种情况很容易判断，因为一般有效页面的信息内容较多，所以抓取的内容中包含的字符串长度要远远大于无效内容的字符串长度，这时可以认为 6 次抓取是有效的。或者已抓取数据量为 10，但其中有 2 次抓取返回了相同的内容，如重复抓取了一个链接，那么有效的次数是 9 次，可以在抓取时动态维护一个已抓取链接的列表，在每次抓取时都与这个列表中的链接数据进行比较，防止重复抓取。此外，连接超时产生的异常次数也可以作为一个抓取参考量。本节设计的监控系统只关注已抓取数据量和有效抓取数量。

（2）抓取数据质量

抓取数据质量是通过更深入地分析有效抓取量进行衡量的。通常抓取到的内容需要进一步清理和抽取，从而获得其中的信息。在一般情况下，信息都是由若干域组成的，如抓取到的图书信息是由书名、作者、出版时间、页数、价格等域组成的。进行抓取之前，会根据需求分析待抓取的内容，确定需要抓取的域，然后将抓取到的内容按这些域进行抽取，形成有效的抓取记录。但往往会出现这样的情况，就是可以抓取到内容并进行抽取，由于内容自身相关信息的缺失，导致抽取出的域是空值，这就需要从另一个角度考虑数据的有效性。比如，对于图书信息的抓取，如果希望用于后续的价格评估，那么价格信息就是一个关键的量，如抓取并抽取出的 10 条数据记录中，有 3 条价格信息是缺失的，那么这 3 条信息就没有用了，这样数据质量就要比缺失 2 条的情况下低，而且价格可能是由作者和页数决定的，在未来进行估价时这两个是首要因素，还要在价格域确定的基础之上确定这两个域的缺失情况。同时，还可能遇到数据域不规范的情况，常见的有写错信息或数值单位不统一等。总之，在设计监控系统，特别是实时监控系统时，需要及时根据已经抓取的数据内容对抓取数据质量进行评估，考虑到要对各域的情况进行分析和统计，为了加快数据检索和查找的速度，可以将抽取出的数据在保存时同时建立索引，这在后面的章节中会有介绍。

本节主要关注监控抓取数量问题，将介绍如何设计一个适合于分布式数据抓取数量监控的平台。通过上面的分析，实现这种监控并不困难，分布式系统中的每个节点只要在本地记录所抓取数据的数量信息，然后实时地将信息发送到监控服务器即可，服务器负责实时收集这些信息，并且提供 Web 接口提供信息的实时报表显示和实时抓取数量图形显示，就可以在需要时利用浏览器访问服务器提供的接口，了解各节点的抓取情况和整个系统的运行情况。

下面简要介绍设计监控系统需要的主要技术。目前可用于设计监控框架的前后端技术有多种，这里提出的各种框架和技术仅供参考。

因为我们使用 Python 作为数据抓取程序的设计语言，各节点在抓取时发送数量信息时也是用 Python 相关的模块，因此 Python 将作为整个系统实现的主要语言。下面介绍使用到的其他相关框架，主要涉及 InfluxBD、Django 和 Grafana。其中，InfluxBD 是一个实时数据库，运行于服务器之上，可用于实时接受各分布式抓取节点传送过来的实时数据；Django 框架主要用于数据报表的展示；Grafana 框架是一个图形界面框架，用来实时地显示各节点和整个系统的抓取情况。

接下来对各框架进行简要说明，介绍相关的技术和监控框架搭建的步骤，最后给出一个使用这些技术的监控框架实例。

8.1　框架说明

1. InfluxDB 实时数据库

网址为 https://influxdata.com/，如图 8-1 所示。InfluxDB 是一种比较流行的实时数据库，可以运行在 Windows 和 Linux 系统上，接收和保存各抓取节点传入的数据信息。

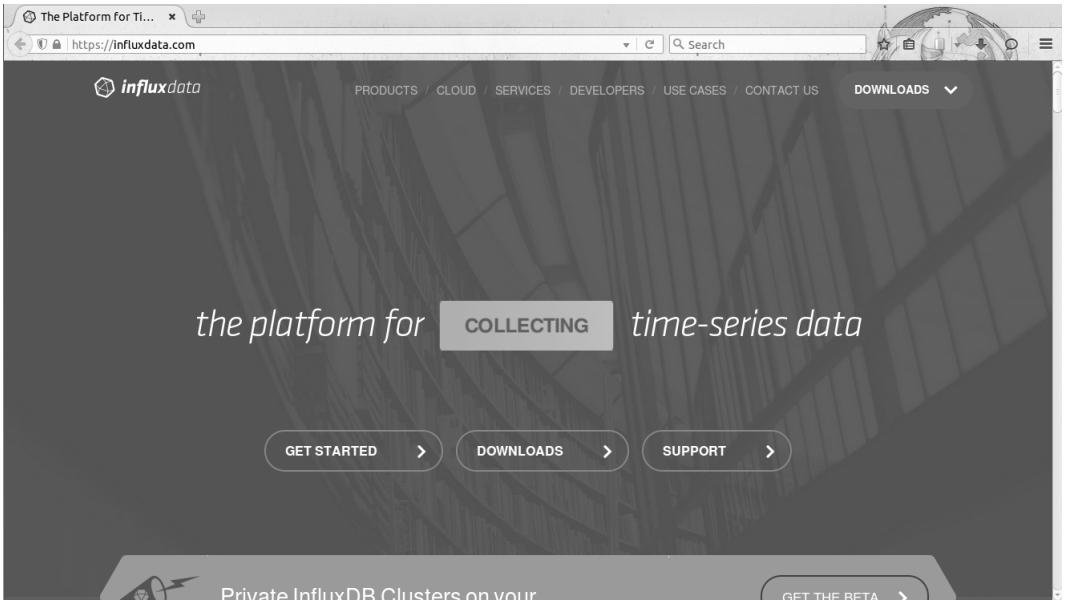


图 8-1　InfluxDB 首页

Python 具有访问 InfluxDB 的模块，可以使用下面的命令安装，即

```
sudo pip3 install influxdb
```

2. Django Web 框架

网址为 https://www.djangoproject.com/，如图 8-2 所示。Django 是一个流行的 Web 开发框架。本节主要使用 Django 开发数据报表展示部分。

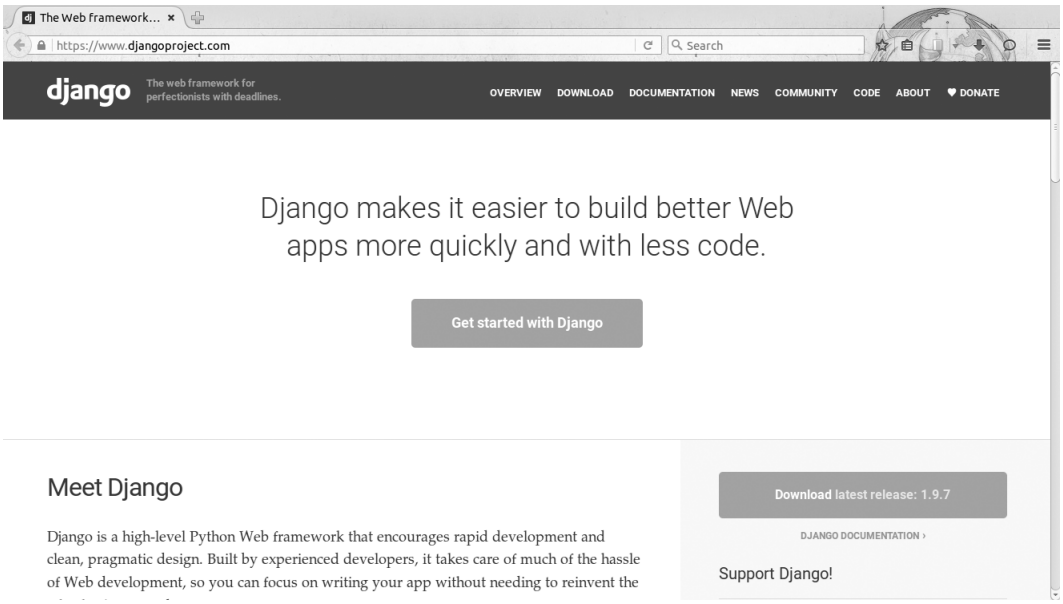


图 8-2　Django 首页

其他的很多 Web 框架也可以选择，如 Flash、Web2py 等。使用框架的目的是可以基于框架的运行模式和使用实例，快速搭建一个可以运行的，能够接受用户请求，并进行响应的系统，但在页面的布局展示细节上，还需要使用 CSS 和 Javascript 等技术。这类技术也有一些成熟易用的框架，在页面展示上，我们使用了 Bootstrap3 布局框架。

3. Grafana 监控框架

网址为 http://grafana.org/，如图 8-3 所示。

Grafana 是一款强大的图表显示框架，特别适用于系统的性能监控，可以实时展示数据量的变化情况。Grafana 能够与 InfluxDB 整合在一起，将实时入库的数据以实时的形式展示出来，配置方便，界面美观。

安装好服务框架后，就可以着手根据需求进行代码的编写了。这些框架的资料非常完备，并且在国内外的技术论坛和博客中有很多 InfluxDB 和 Grafana 结合的实例可供参考。代

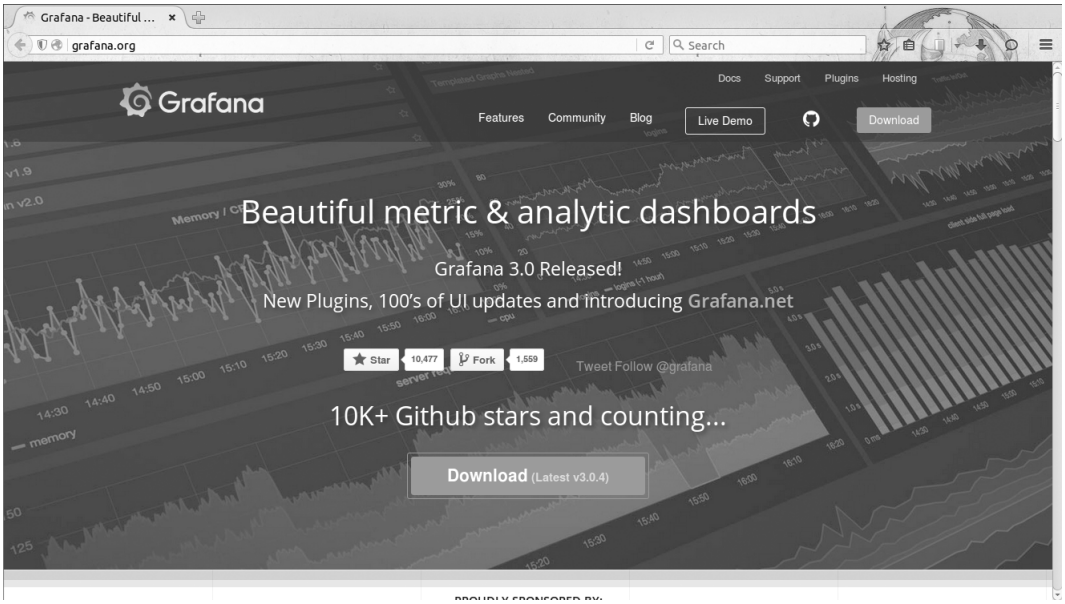


图 8-3　Grafana

码的逻辑并不复杂，各节点不断有数量信息发送到 InfluxDB 服务器。Django 提供的服务可以进行页面报表展示，根据我们不同的报表请求，利用 Python 与 InfluxDB 交互的模块向 InfluxDB服务器发送请求，并将返回的结果在页面中展示出来。Grafana 直接就提供和 InfluxDB 的接口，可以在界面配置连接信息，实时获得 InfluxDB 的数据。因为篇幅所限，这里不提供代码实现的细节，感兴趣的读者可以自己查找资料，来实现符合自己要求的实时监控平台。这个过程中涉及到流行框架的学习、前后端设计、实施及交互的问题，具有较强的综合性，是一个十分全面的学习案例。

8.2　监控系统实例

下面给出一个分布式房地产信息抓取平台的实时监控系统。首先介绍一下抓取的主要内容。房地产数据主要包括各类网签成交数据、新房信息、二手房挂牌房源信息、二手房成交信息、各种租赁信息等。以二手房交易为例。为了及时准确地获得全国二手房市场的房源信息，首先应确定信息来源网站。在实施时选取了几个数据量大的、信息权威的房地产信息发布平台作为信息来源，实现对其发布的房源信息进行实时抓取。然后，对各个平台的数据进行分析，总结各网站信息组织的共同特点，如各类平台的信息组织形式均为“信息列表 -> 详细信息”的层级组织方式，同时在分析时也要确定抓取的信息字段。接下来就可以着手编写代码并在本机或小范围的分布式环境下进行测试，根据测试结果对代码

进行进一步的修改和调试。最后，将整体框架部署在实际分布式环境中，部署时，服务器选择的是一台 128G 内存的 Windows 服务器，抓取端选择了 12 个具有独立 IP 地址的阿里云端。

下面就是由 InfluxDB、Django 和 Grafana 实现的抓取信息实时监控系统，可以利用该系统方便地实时监控整个分布式环境的数据抓取情况，及时获得各种数据指标，掌握系统运行的情况。图 8-4 为监控系统首页。

图 8-4　监控系统首页

通过页面左边菜单入口可以对各抓取端的工作性能和工作情况从不同角度进行查询。通过左边最下端的“实时查看”可以进入数据指标图形统计界面。图 8-5 ~ 图 8-8 分别给出了抓取数量观测、错误统计、数量报表和抓取爬虫情况等信息。

图 8-5　抓取数量观测

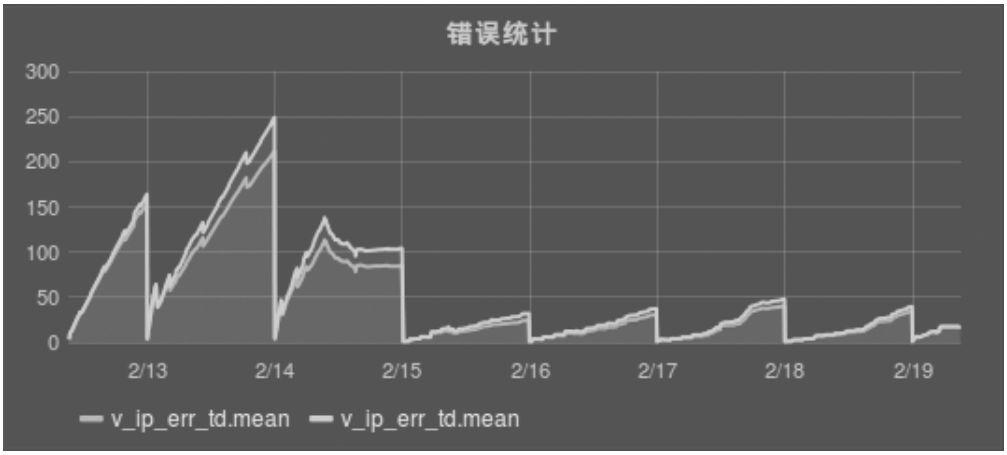


图 8-6　错误统计

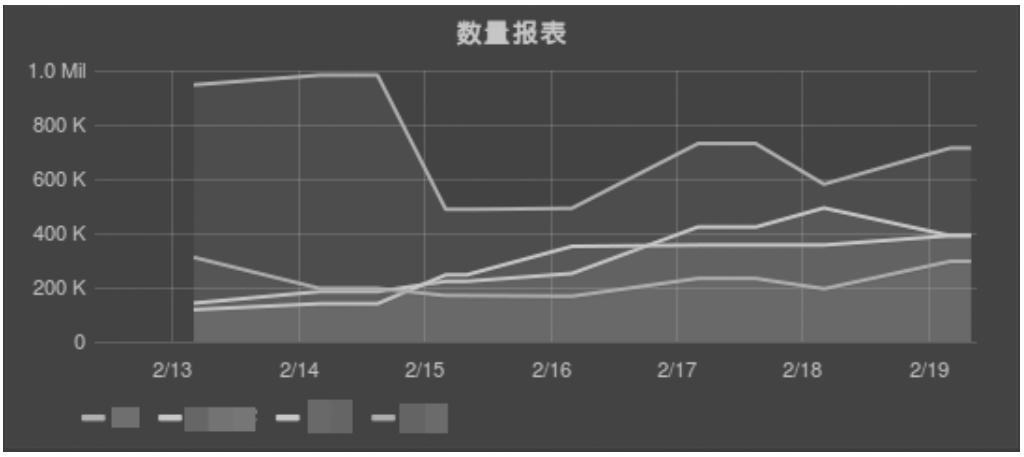


图 8-7　数量报表

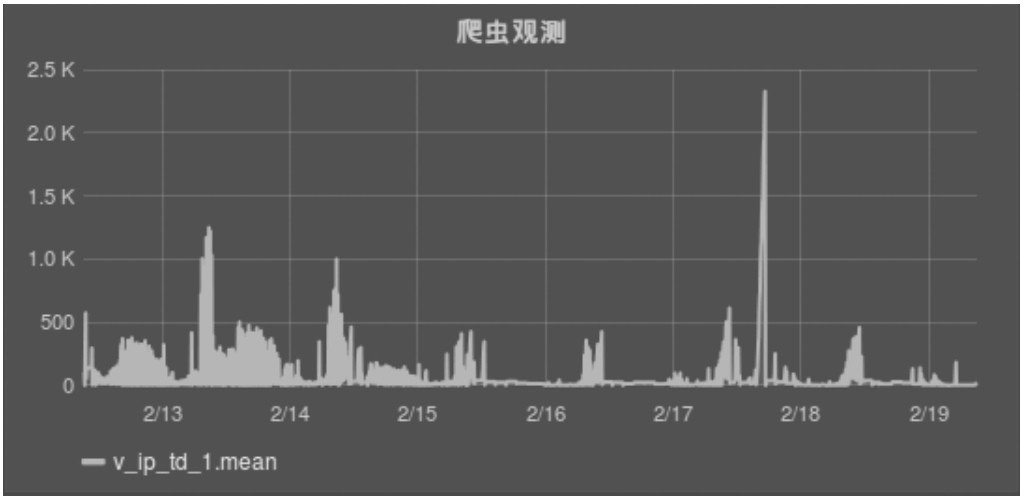


图 8-8　抓取爬虫情况

图 8-9 给出了各爬虫抓取情况。横轴代表时间，纵轴代表相应时间的抓取量。从图中可以看出，各抓取端的负载是均衡的，这从图形的模式可以看出，图形的形状相似。此外，设计的分布式抓取系统采取的是实时抓取策略，通过各图的分析可以看出，每天在 10 ~ 11 点信息的发布量达到一个峰值，这与实际情况是吻合的。

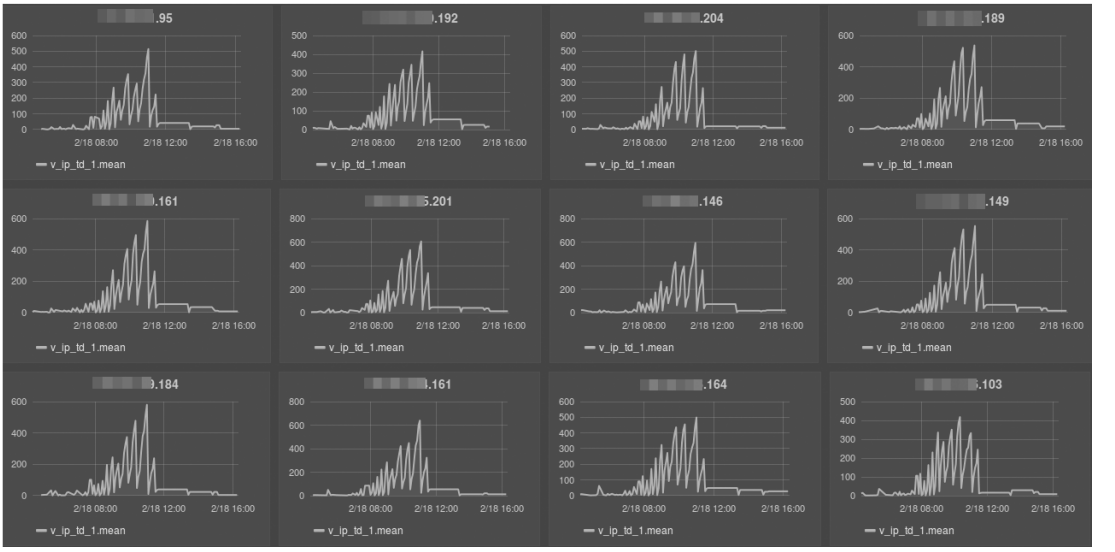


图 8-9　各爬虫抓取情况

实战

根据本章介绍的框架或其他的类似框架搭建和部署一个实时监控抓取数据量的平台。

9 第9章 拥抱大数据

9.1 Hadoop 生态圈

执行一次性的抓取，抓取到的数据量往往较小，可以存放在文本文件、Excel 中供使用和分析。如果数据量较大，则可以将数据存放在 Mysql、Oracle、PostgreSQL 等关系型数据库中，也可以存放在 MongoDB 非关系型数据库中。各类数据库都有强大的索引能力和方便的数据操作接口，可以极大地提高在未来对数据的分析和查找使用的工作效率。

如果抓取的数据量特别大，如针对某类信息进行有计划地周期性抓取而积累起来的数据，则会在存储和分析上给我们带来极大的挑战，此时就可以考虑大数据技术提供的解决方案。当前以 Apache Hadoop 为核心的大数据生态圈为大数据的存储和分析提供了强大的支持，可以将抓取到的大量数据存入 Hadoop 支持的分布式文件系统中，利用 Hadoop 提供的各类数据分析工具进行处理和分析，在 Hadoop 的官方网站 http://hadoop.apache.org/列出了Hadoop相关的主要 Apache 工程，如图 9-1 所示。

Other Hadoop-related projects at Apache include:

- **Ambari™**: A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heatmaps and ability to view MapReduce, Pig and Hive applications visually alongwith features to diagnose their performance characteristics in a user-friendly manner.
- **Avro™**: A data serialization system.
- **Cassandra™**: A scalable multi-master database with no single points of failure.
- **Chukwa™**: A data collection system for managing large distributed systems.
- **HBase™**: A scalable, distributed database that supports structured data storage for large tables.
- **Hive™**: A data warehouse infrastructure that provides data summarization and ad hoc querying.
- **Mahout™**: A Scalable machine learning and data mining library.
- **Pig™**: A high-level data-flow language and execution framework for parallel computation.
- **Spark™**: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- **Tez™**: A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases. Tez is being adopted by Hive™, Pig™ and other frameworks in the Hadoop ecosystem, and also by other commercial software (e.g. ETL tools), to replace Hadoop™ MapReduce as the underlying execution engine.
- **ZooKeeper™**: A high-performance coordination service for distributed applications.

图 9-1　Hadoop 相关的主要 Apache 工程

除此之外，围绕着大数据处理 Apache 还有很多其他的工程。为了方便使用，很多大数据服务商将这些工程进行了整合，基于这些整合方案，在使用和部署时不必手工下载、安装、配置各种工程，可直接使用服务商提供的安装管理包，按照提示进行安装配置即可。比较有代表性的大数据服务商是 Cloudera。Cloudera 不但提供了大数据应用安装和管理的整体解决方案，还推出了图形化的界面管理工具。本节的内容将基于 Cloudera 提供的方案，通过模仿和修改 Cloudera 提供的成熟应用实例达到入门的目的。在大数据应用领域，各种技术层出不穷，通过可运行的案例了解一些技术的基本使用方法可以快速地入门，当学习完一个实例再回头看时，不仅加深了对相关概念的理解，还为进一步的应用打下基础。

结合本书的数据抓取实例，本节介绍一个实时抓取数据的应用，通过实时索引的方式保存，利用索引快速搜索出需要的抓取信息，也可以在实时抓取的同时，利用索引对某些关键的字段进行快速实时的分析统计，从而对抓取质量进行实时的评估。我们将要讲述的实时抓取和索引的实例与实时的日志收集和索引的工作模式是一致的，而后者在行业内已经具有了十分成熟的解决方案。本节采用 Apache Flume 和 Apache Solr 相结合的解决方案，并参考 Cloudera 提供的实时日志收集和索引实例进行修改和配置。Cloudera 已经整合好了这两个环境，但是 Flume 和 Solr 也可以分别单独安装。图 9-2 和图 9-3 分别是它们的工程首页，在首页中进入下载页面，按文档单独下载和安装即可。

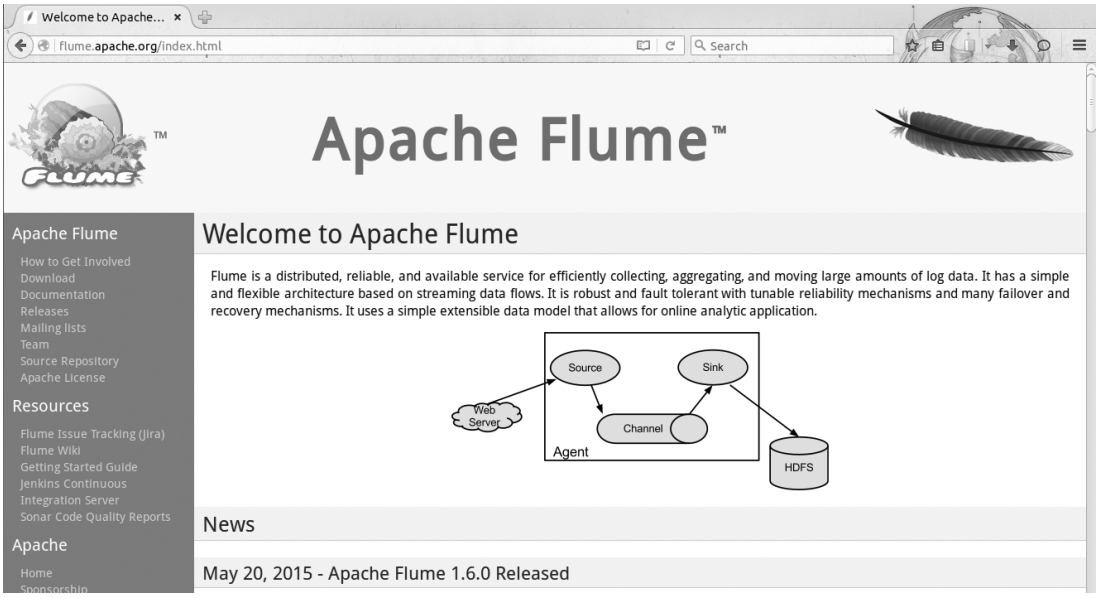


图 9-2　Flume 工程首页

在开发大数据分布式应用时，通常先在一台机器上进行开发和测试，当在一台机器上跑通后，才会将其部署在实际的分布式环境中。下面就采用这种开发形式开发我们需要的应用。

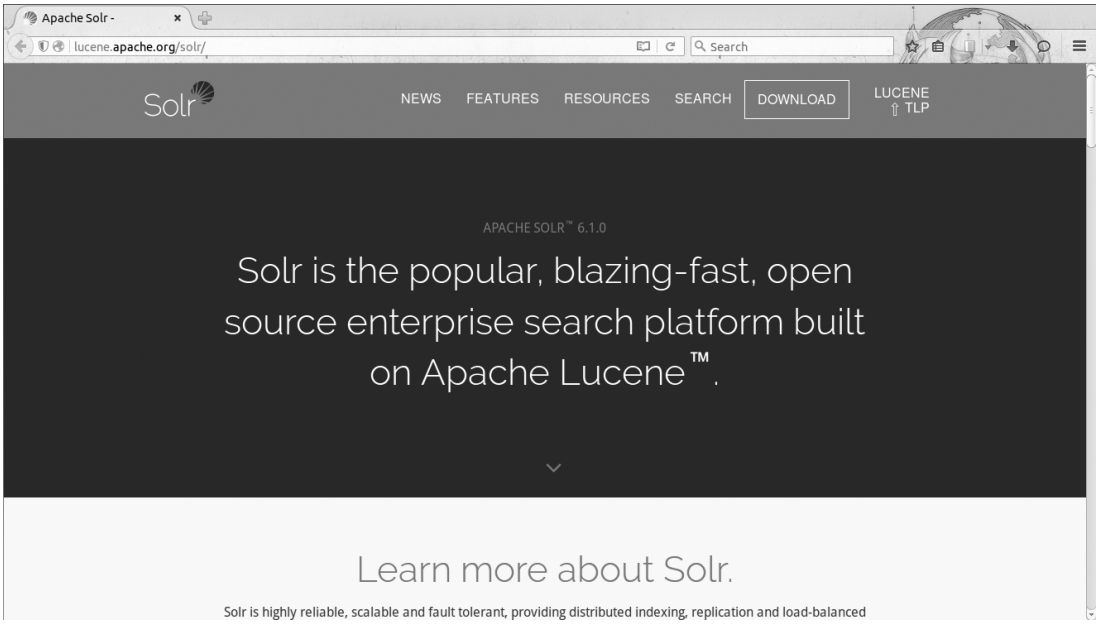


图 9-3　Solr 工程首页

9.2　Cloudera 环境搭建

首先搭建环境，在开发阶段，可以在单机搭建环境安装 Flume 和 Solr，即在这两个工程的官方网站下载相关的文件，根据它们各自的文档进行配置。还有一种更为简便的方式，就是直接使用 Cloudera 提供的，可在单机环境使用的已经配置好各种大数据服务环境的 docker 镜像和虚拟机镜像。

1. docker 镜像的使用

可以按照 Cloudera 官方提供的教程进行镜像的下载和使用，教程的网址为 https://hub.docker.com/r/cloudera/quickstart/。

（1）安装 docker

```
sudo apt-get install docker.io
```

（2）从 Docker Hub 导入 Cloudera QuickStart 镜像

```
sudo docker pull cloudera/quickstart:latest
```

这个过程耗时较长，可能会由于网络的原因而失败，若失败，则可以重新运行命令导入。

（3）运行使用镜像的容器

导入成功后，可以使用命令 docker images 查看，如图 9-4 所示。

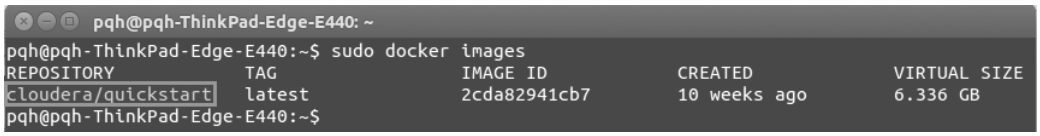


图 9-4　镜像查看

图中，框起的“cloudera/quickstart”就是镜像的名字，可以根据镜像名字运行容器，命令为

```
sudo docker run --hostname=quickstart.cloudera --privileged=true -t -i -p 8888:6789 cloudera/quickstart/usr/bin/docker-quickstart
```

该命令的参数可查看 docker 教程或 Cloudera 官方网站的说明，注意命令中黑体加粗的部分，-p 8888：6789 表示将 Hue 的端口 8888 映射到主机的 6789，cloudera/quickstart 就是使用 docker images 查出的镜像名。运行命令，如图 9-5 所示。

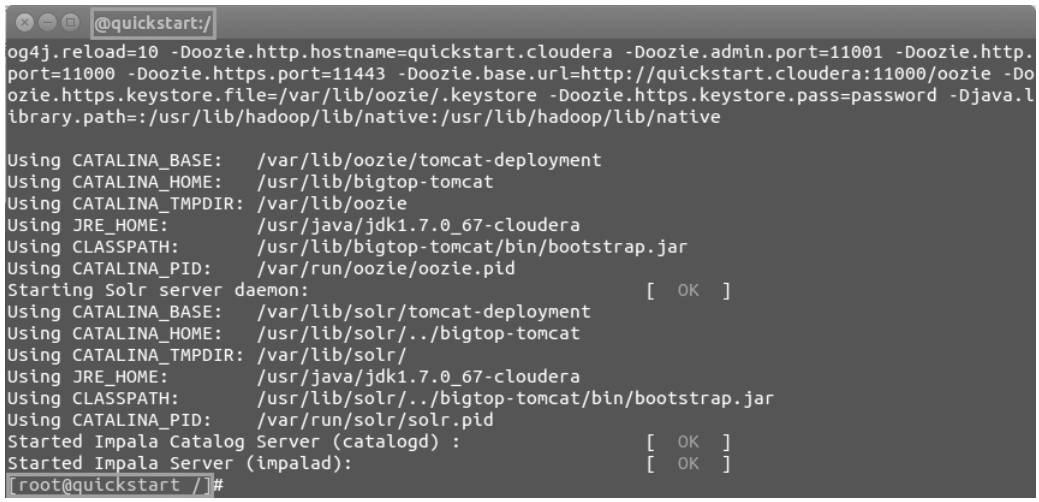


图 9-5　启动容器

启动成功后，可以访问主机的 6789 端口，访问容器内 Hue 端口 8888 提供的服务。在主机打开浏览器，输入 http://127.0.0.1:6789/，会显示登录界面，如图 9-6 所示。

默认的用户名和密码都是 cloudera，登录后的界面如图 9-7 所示。

可以在这种方式下进行大数据相关技术的学习和实验，在 docker 启动容器的 shell 中运行程序，编辑配置文件和编程，也可以在 Web 页面中进行各种数据的操作。

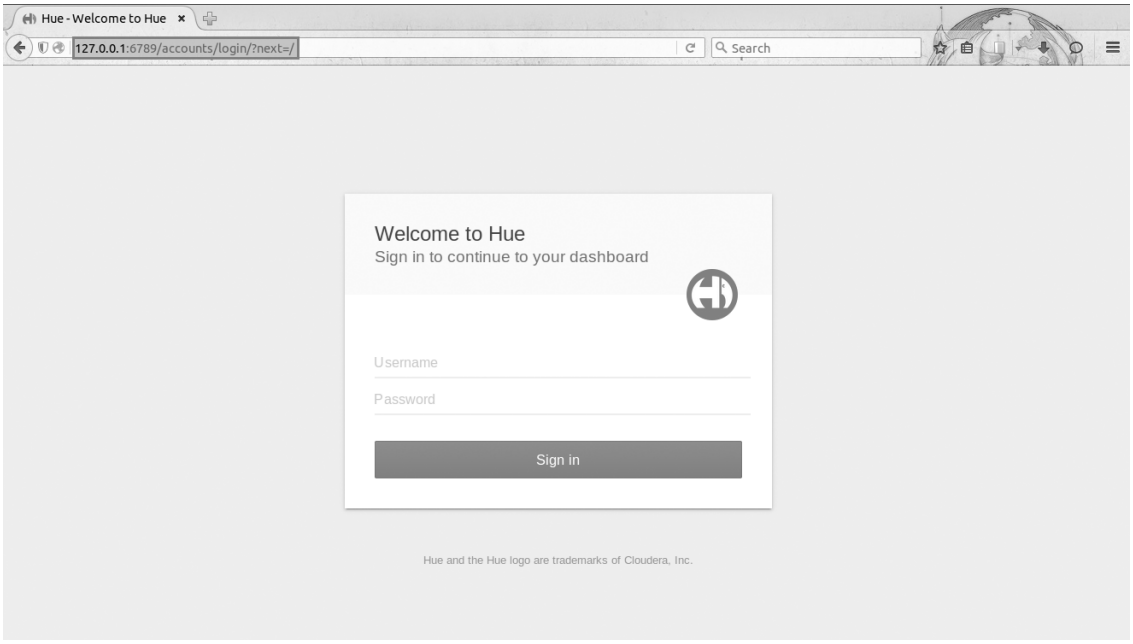


图 9-6　登录界面

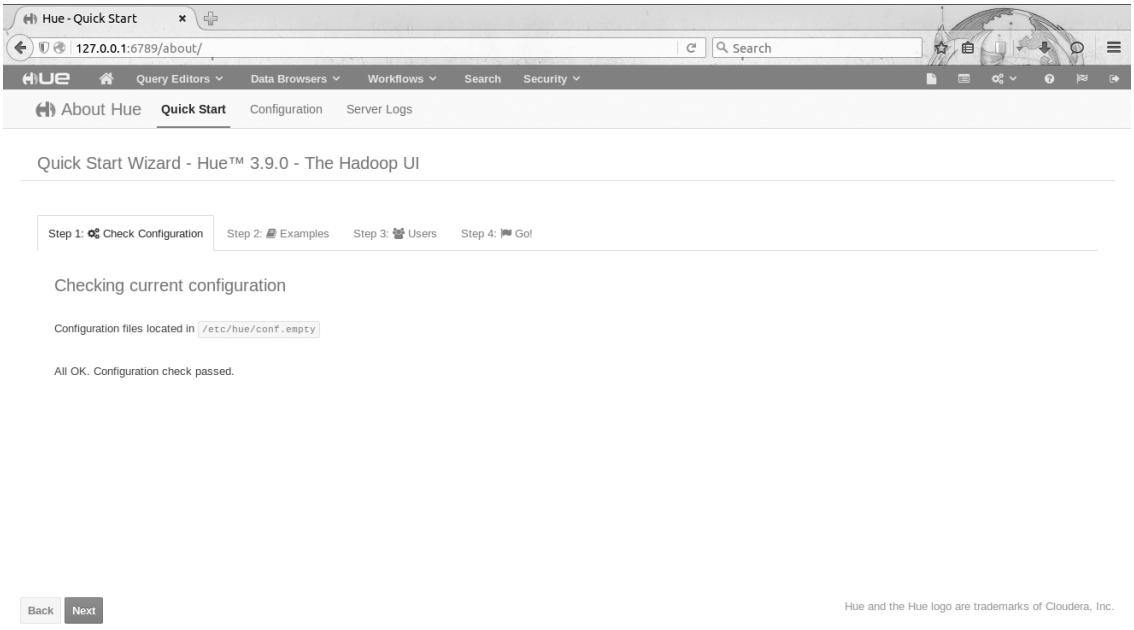


图 9-7　登录后的界面

另一种方式是使用 Cloudera 提供的虚拟机。本节将主要使用 Cloudera 提供的 Vmware 虚拟机镜像。下面介绍这种方式。

2. 使用 Vmware 镜像

打开网页 https://www.cloudera.com/downloads.html，如图 9-8 所示。

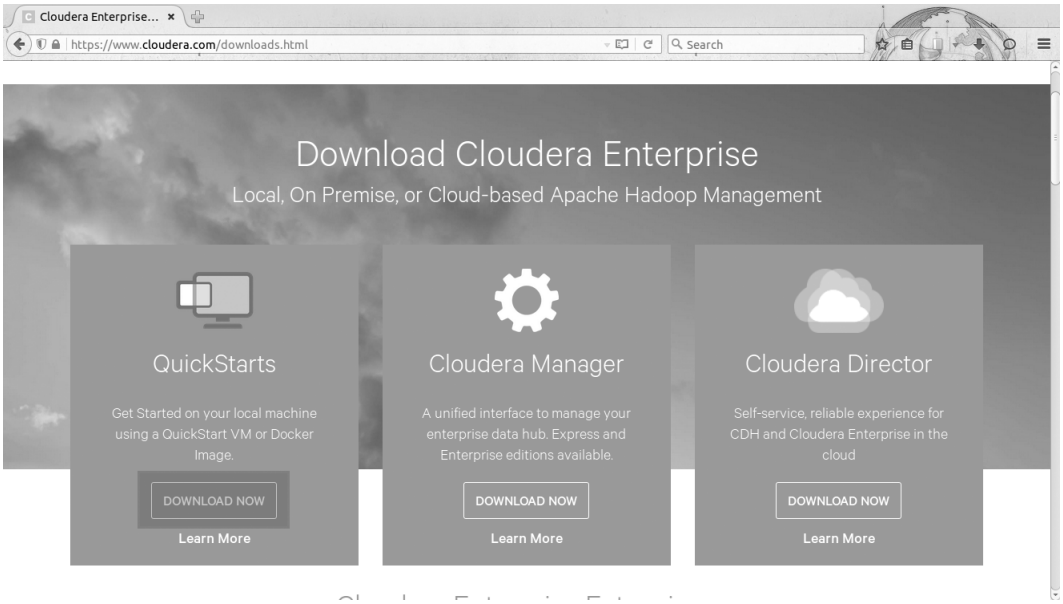


图 9-8　Cloudera 首页

在首页单击下载，进入下载页面，如图 9-9 所示。

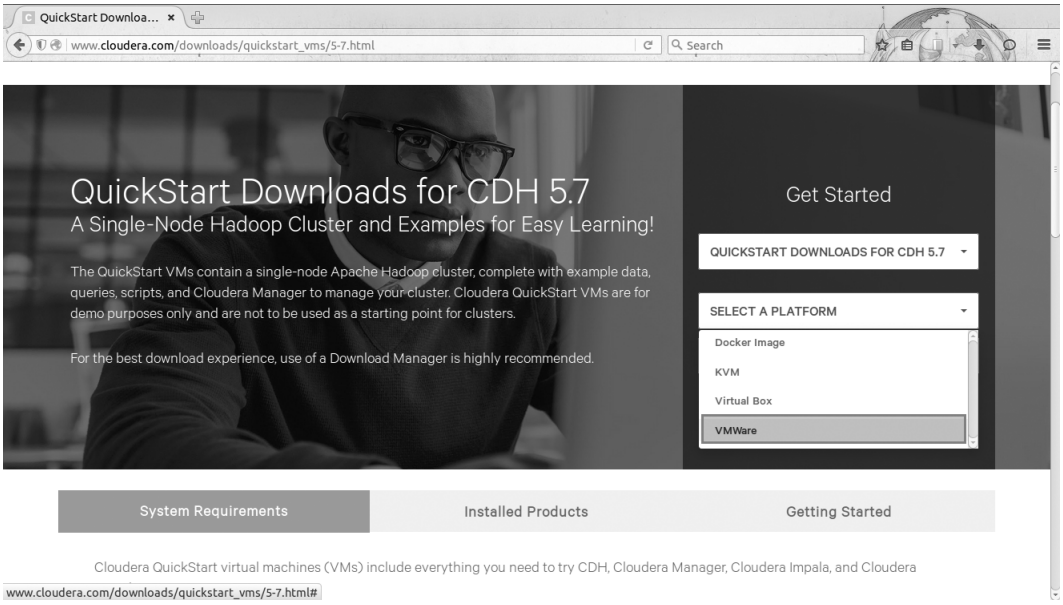


图 9-9　Cloudera 下载页面

选择后，单击选择框下的“DOWNLOAD NOW”即可，在下载前需要填写一些信息，按要求填写即可。

下载后的文件名为 cloudera – quickstart – vm – 5. 7. 0 – 0 – vmware. zip，约 5. 1G 左右，直接解压即可。在解压后的文件夹中找到 cloudera – quickstart – vm – 5. 7. 0 – 0 – vmware. vmx 文件，如果已安装了 Vmware 工具，则直接双击即可启动虚拟机，在启动界面单击，如图 9–10所示。启动后的系统如图 9–11 所示，为 Centos 操作系统。

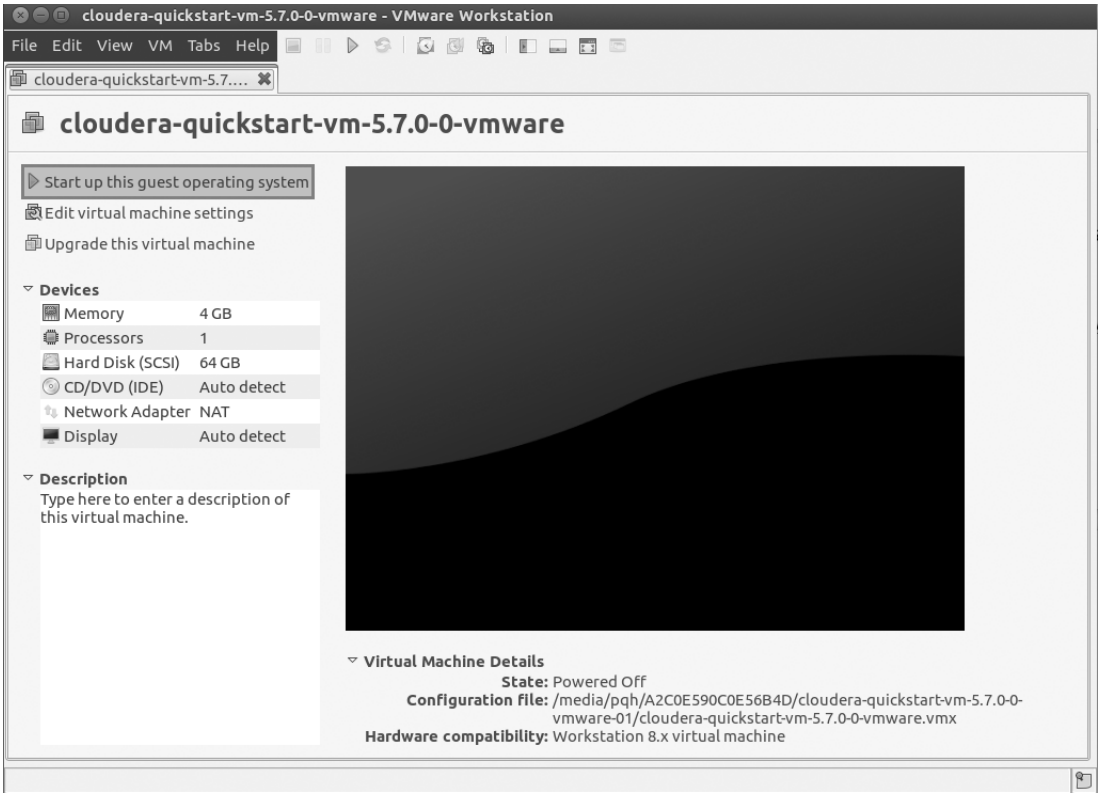


图 9–10　打开虚拟机并启动系统

如果没有安装 Vmware，就需要先安装工具。本节将使用的安装文件是 VMware – Workstation – Full – 12. 1. 0 – 3272444. x86_64. bundle，别的版本也可以选择，下载完成后，需要在安装文件的目录中运行

```
chmod a + x VMware – Workstation – Full – 12. 1. 0 – 3272444. x86_64. bundle
```

将该文件改为可执行文件，然后使用命令

```
sudo . /VMware – Workstation – Full – 12. 1. 0 – 3272444. x86_64. bundle
```

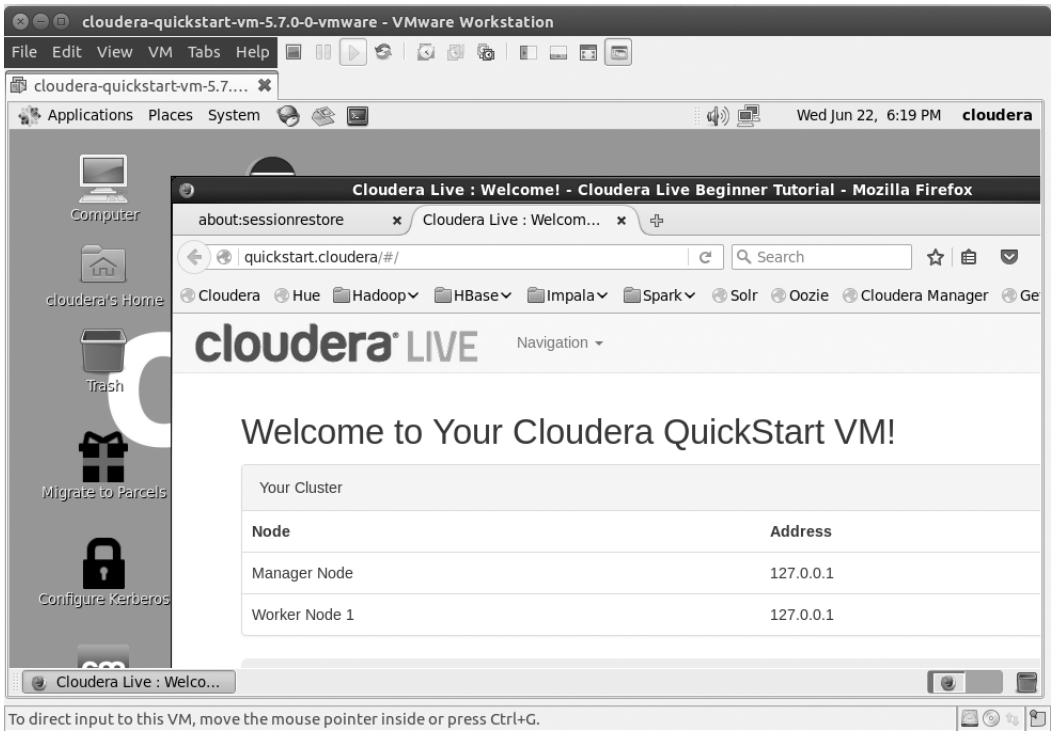


图 9-11　启动后的系统

安装，在安装过程中会出现“下一步”这样的提示界面，按提示一步步安装即可。安装完成后，可启动 Vmware，通过打开“File”菜单，单击“Open…”，在相应的文件夹下打开 cloudera - quickstart - vm - 5. 7. 0 - 0 - vmware. vmx。

3. 学习并修改和配置实例

前面提到过，本节介绍实时抓取的实时索引并保存的实例与实时日志收集系统的工作模式是极其类似的。cloudera - quickstart 镜像中有一个实时日志收集系统实例，可以在这个实例上加以修改来完成我们需要的功能。

该实例为 Tutorial Exercise 4，有一个模拟不断产生日志的程序 start_logs，有一个接收产生日志内容的 access. log，可以使用 tail_logs 观察产生的日志，stop_logs 可以终止 start_logs 程序、终止日志的产生。在 Cloudera 虚拟机环境中，start_logs、tail_logs 及 stop_logs 已经被配置好，都可以在 shell 中直接启动。该实例中两个使用到的工程应用是 Flume 和 Solr，下面简要介绍一下它们的基本作用。Flume 负责监控 access. log 文件，使用的方式是通过下面的命令，即

```
tail - F/opt/gen_logs/logs/access. log
```

其中，/opt/gen_logs/logs/是实例中 access. log 文件保存的目录，tail – F 是系统命令，可以跟踪文件的改变，通过跟踪 access. log 文件，Flume 可以将跟踪获得的文件变化内容作为自己信息的源头，即 Flume 中的 source，然后将获得的信息利用 Morphline 逐条进行提取，提取到的内容传递给 Solr 进行索引和保存。这里 Solr 对于 Flume 来说就是 sink。在 Solr 成功索引并保存信息后，未来就可以利用 Solr 提供的接口对索引数据进行查询和分析。Flume 与 Solr 与其他大数据工具使用的方式一样，使用前都需要根据需要对配置文件进行相关参数的设定，然后启动。启动后，无论是以应用程序方式还是以后台服务方式，这些工具都会根据指定的参数执行任务。Flume 和 Solr 的参数十分丰富，可以根据不同的使用情况进行配置。我们的实例将在 Tutorial Exercise 4 配置文件的基础上进行修改。若想完成其他的功能，则可参考各自的文档。

（1）需要安装的软件

如果使用前面安装的 Vmware 版本，那么在虚拟机启动之后，利用虚拟操作系统中的火狐浏览器就可以直接上网了，但打开网页后会出现乱码，因此需要安装中文支持，同时在抓取数据、索引数据和存储数据时也要处理中文信息，安装命令为

```
sudo yum groupinstall chinese – support
```

此时，打开网页就会看到中文显示正常了。

虚拟系统中默认的 Python 版本是 Python2. 6，为了处理中文方便，最好安装 3 以上版本的 Python，可以安装 Python3. 4，安装时，可以将 Python 的软件包下载并解压，然后在解压目录中执行 Linux 安装软件时常用的 3 条命令，即 configure、make 和 make install。

首先使用 wget 下载 Python 的压缩文件，打开 Centos 的终端，可以单击如图 9–12 所示的终端图标打开。

图 9–12　终端图标

在默认的情况下，因为 Vmware 支持虚拟操作系统共享主机的网络，所以在打开的终端中可以利用 cd 进入希望保存下载文件的目录，然后运行下面的命令即可。

```
$ wget https://www. python. org/ftp/python/3. 4. 4/Python – 3. 4. 4. tgz
```

下载完成后，可以使用 tar 命令解压，进入解压后的文件夹进行配置和安装，命令为

```
$ tar zxvf Python - 3. 4. 4. tgz
$ cd Python - 3. 4. 4
$ ./configure
$ make
$ sudo make install
```

安装完成后，需要安装 requests 和 BeautifulSoup，命令分别为

```
$ sudo pip3 install requests
$ sudo pip3 install bs4
```

安装完成后，就可以编写抓取程序了。下面给出程序代码，然后给出实现实时信息抓取索引的整个流程。新建一个文件夹，在 Home 目录下建立 crawl_indexing 目录，代码及 Flume 和 Solr 相关的一些配置文件都会放在这个文件夹中。

（2）编写抓取程序

本节使用的抓取程序是在前面几节介绍过的，即在 www. phei. com. cn 上抓取图书信息列表页，然后在列表页抽取出书名和详细信息地址的抓取程序做了一些改动，将程序变为一个无限循环，循环抓取计算机类图书的前 4 页图书信息列表内容，抽取出的信息会写入到一个文件 access. txt 中，用这个过程来模拟有信息不断到来的情况。代码如下，文件名为 crawler. py。

```
    import requests
    from bs4 import BeautifulSoup
    import time

    def format_str(s):
    return s. replace("\n","". replace("","". replace("\t","")

def get_urls_in_pages(from_page_num, to_page_num):
    global cnt
    urls = []
    search_word = 计算机
    url_part_1 = http://www. phei. com. cn/module/goods/ \
                 ' searchkey. jsp? Page =
    url_part_2 = &Page = 2&searchKey =
    for i in range(from_page_num, to_page_num + 1):
        urls. append(url_part_1
                     + str(i) +
```

```
                    url_part_2 + search_word)
    all_href_list = [ ]
    for url in urls:
        print(url)
        resp = requests. get(url)
        bs = BeautifulSoup(resp. text)
        a_list = bs. find_all('a')
        needed_list = [ ]
        for a in a_list:
            if 'href' in a. attrs:
                href_val = a['href']
                title = a. text
                if 'bookid' in href_val and 'shopcar0. jsp' \
                        not in href_val and title ! ='':
                    if [title, href_val] not in needed_list:
                        needed_list. append([format_str(title),
                                             format_str(href_val)])
        all_href_list += needed_list
    all_href_file = open('access. txt', a+')
    for href in all_href_list:
        tstamp = str(int(time. time()))
        all_href_file. write('\t'. join(href) + '\t' + tstamp + '\n')
        all_href_file. flush()
        print('write', '\t'. join(href) + '\t' + tstamp)
        time. sleep(2)
    all_href_file. close()
    print(from_page_num,to_page_num,len(all_href_list))

def crawl():
    while True:
        get_urls_in_pages(1,5)
        t1 = time. time()
        t2 = time. time()
        print('使用时间',t2 - t1)

if __name__ ='__main__':
    crawl()
```

代码的功能如前所述，注意在向文件中写入时增加的语句

```
all_href_file.flush()
```

这样就可以将缓存中的内容及时地写入到文件中，同时在写入的内容中增加了 tstamp 字段。该字段的内容为

```
tstamp = str(int(time.time()))
```

因为在重复运行的过程中会反复出现相同的信息，所以使用时间字符串可以将这些信息区分开，同时也可以作为记录的 ID。

下面运行这段代码，打开终端，进入当前程序所在目录 crawl_indexing，执行代码，如图 9-13 所示。

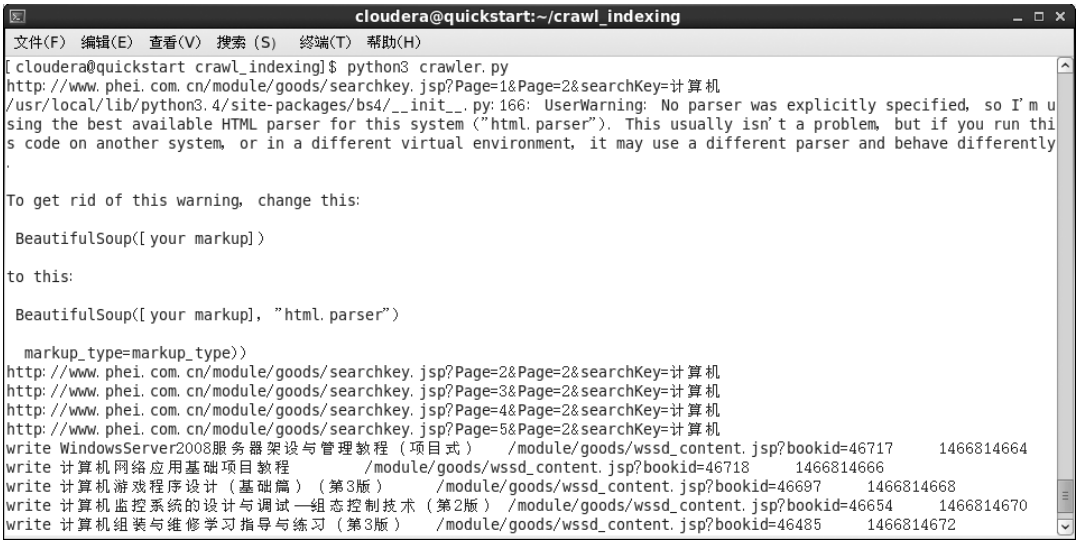


图 9-13　执行程序

可见，程序可以正常运行。图 9-14 为 access.txt 的内容。

现在就可以考虑为抓取的内容创建索引了，需要配置 Solr 的相关参数。下面描述这一过程。

（3）Solr 建立索引

首先打开 shell 进入目录 crawl_indexing，在该目录下运行命令

```
solrctl --zk quickstart:2181/solr instancedir --generate solr_configs
```

这将在 crawl_indexing 目录下生成 solr_configs 文件夹，如图 9-15 所示。

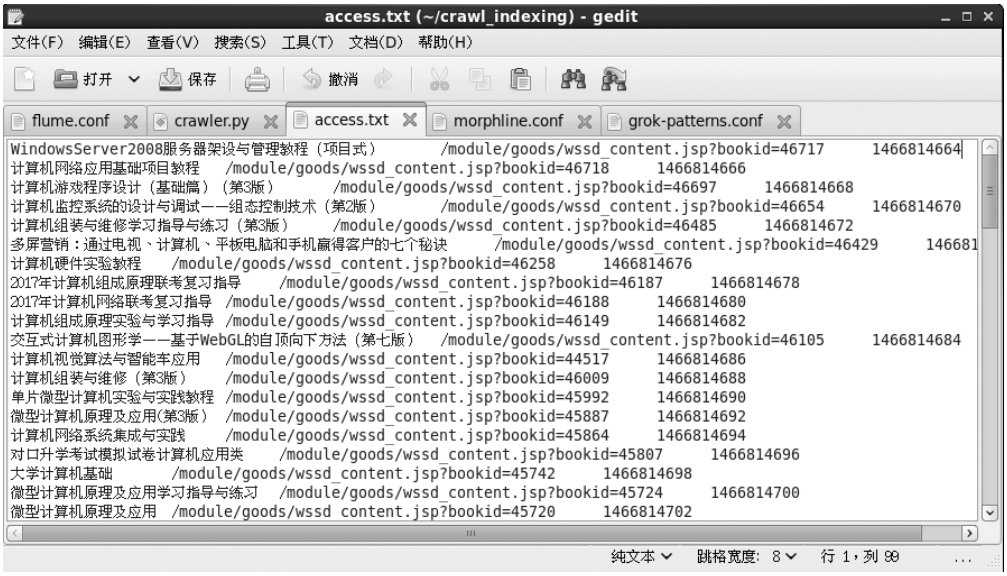


图 9-14　access. txt 的内容

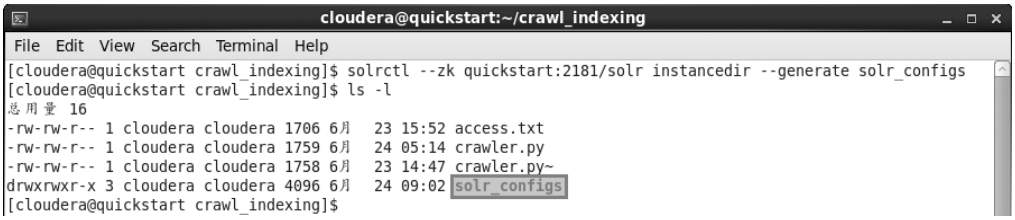


图 9-15　生成配置文件夹

在该文件夹下还有一个文件夹 conf，在 conf 文件夹下是 Solr 所需要的配置文件，这里只需要编辑和修改 schema. xml 文件，在该文件中针对抓取到的信息设置相应的索引域。可以使用 Centos 中的 gedit 编辑器打开 schema. xml 文件，通过搜索“ < fields > ”定位到设置域属性的位置，在“ < fields > …… </fields > ”内就可以设置域信息。抓取信息的格式分为书名、url 相对地址、时间戳字符串三部分，可以在“ < fields > …… </fields > ”中加入下面的定义，加入时可以参考在该范围内其他已经定义好的域，这里设书名、url 相对地址名称分别为 book_name、url_rel。为简单起见，它们的 type 也都设为 text_general，时间戳可以作为 ID，而 schema. xml 默认已经设置了这个域，设置如图 9-16 所示。

配置完成后，在 shell 中重新回到 crawl_ indexing 目录，可以使用下面的命令将配置上传。

```
solrctl -- zk quickstart:2181/solr instancedir -- create info . /solr_configs
```

如图 9-17 所示。

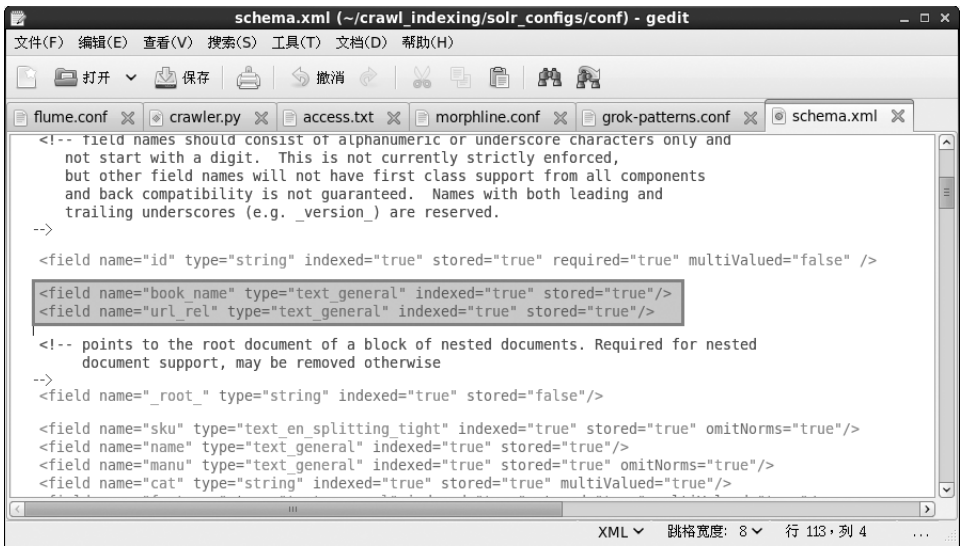


图 9-16　schema. xml 内容修改

```
[cloudera@quickstart crawl_indexing]$ solrctl --zk quickstart:2181/solr instancedir --create info ./solr_confi
gs
Uploading configs from ./solr_configs/conf to quickstart:2181/solr. This may take up to a minute.
[cloudera@quickstart crawl_indexing]$ _
```

图 9-17　上传配置

接下来使用下面的命令在 Solr 中创建集合（collection）info。

```
solrctl -- zk quickstart:2181/solr collection -- create info  - s 1
```

如图 9-18 所示。

```
[cloudera@quickstart crawl_indexing]$ solrctl --zk quickstart:2181/solr collection --create info -s 1
[cloudera@quickstart crawl_indexing]$ _
```

图 9-18　创建集合 info

现在使用浏览器可以看到创建的索引情况，利用 Hue 控制界面进入 Solr 相关的页面。在前面演示 docker 映像的使用方式时给出了访问 Hue 的方式，在虚拟环境中可以直接打开 Hue，在浏览器内输入 http://127. 0. 0. 1:8888 即可。如果需要用户名和密码，则都是 Cloudera。进入后，在顶部导航区单击 search，进入 http://127. 0. 0. 1:8888/search/界面，如图 9-19 所示。

单击右上角的“Indexes”，进入 http://127. 0. 0. 1:8888/indexer/，在该页中可以看到刚刚建立的 info 集合，如图 9-20 所示。

单击 info，可以进入 info 相关索引域的详细信息页，如图 9-21 所示。

单击右上角的“Search”可以进入 info 的搜索页，如图 9-22 所示。

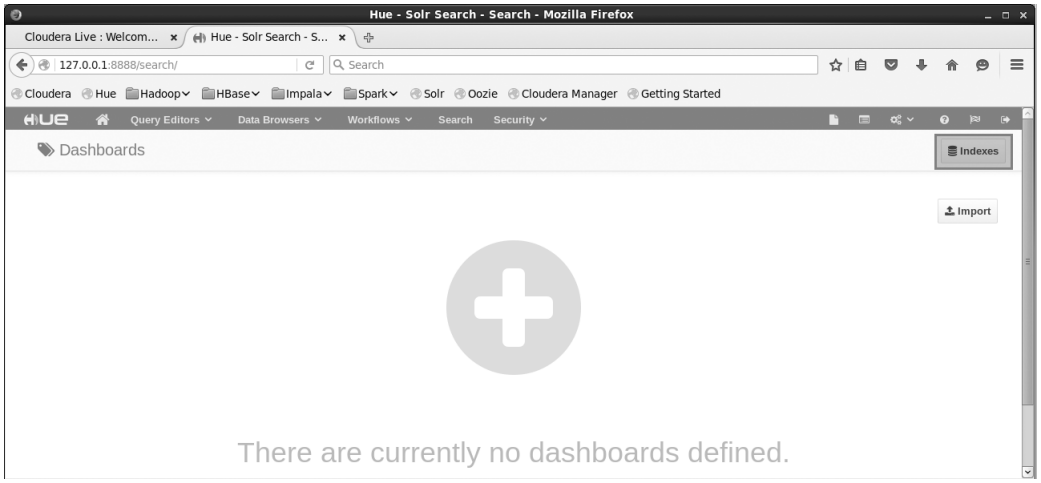


图 9-19　进入 search 界面

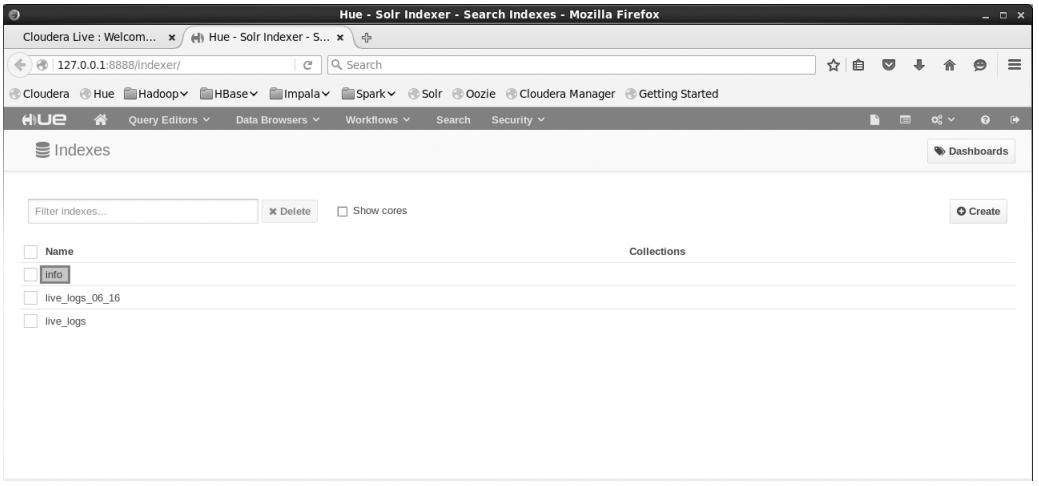


图 9-20　可以看到刚刚建立的 info 集合

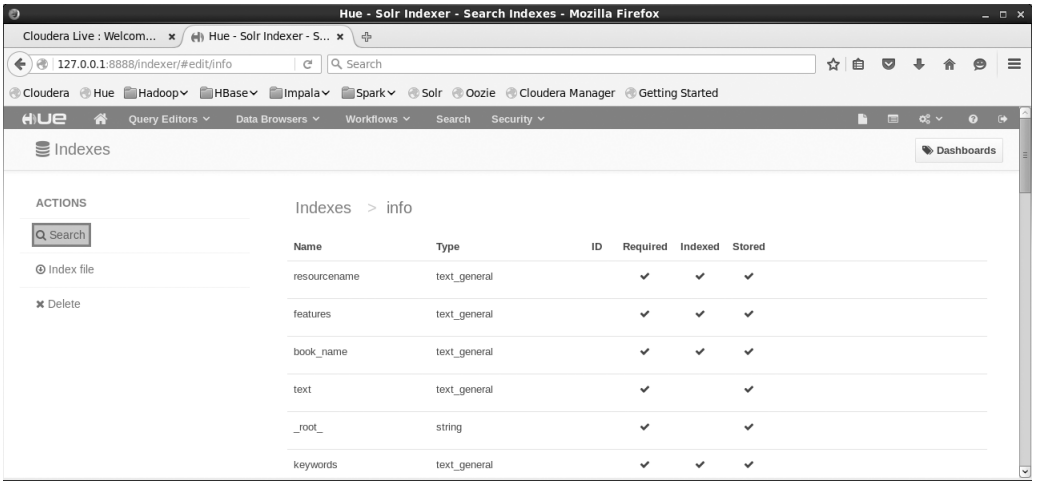


图 9-21　info 相关索引域的详细信息页

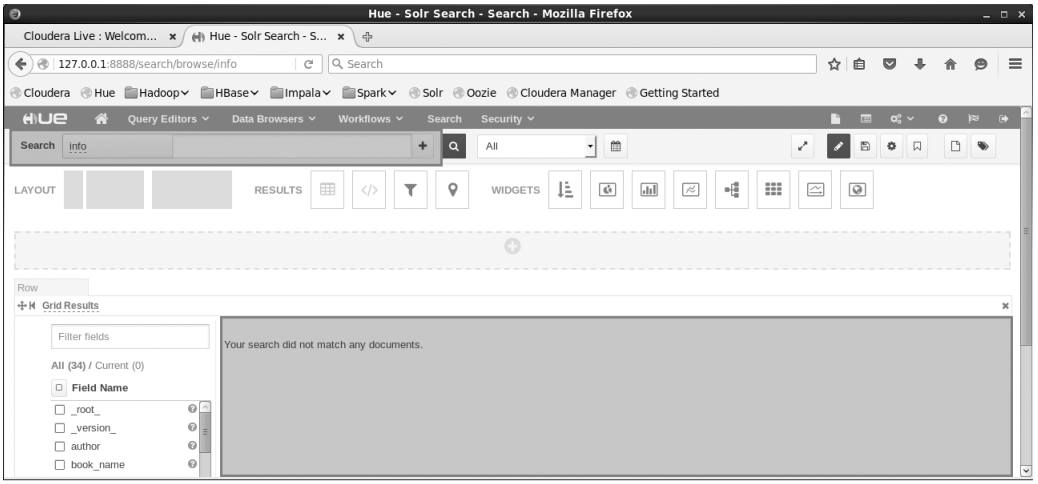


图 9-22　info 目前没有数据

在左上角的方框中表示未来将在这个位置输入查询条件，对索引进行检索。右下角的方框将显示符合查询条件的结果，因为当前还没有数据进入，所以方框内没有内容。下面将 Flume 配置完成后启动抓取程序，抓取到的数据可以通过 Flume 抽取和过滤传入到 Solr 中。

（4）Flume 的配置方法

对 Flume 进行配置时，涉及 Flume 自身的 source、sink 和 channel 配置，因为使用到了 Morphline，因此也需要考虑它的配置。

在 crawl_indexing 下新建一个目录“flume_conf”，因为是在 Tutorial Exercise 4 的基础上进行设计的，所以可以将实例中涉及的两个配置文件 flume. conf 和 morphline. conf 复制过来，然后在此基础上进行简单修改即可。这两个文件在/opt/examples/flume/conf/目录下，同时也需要将 flume - env. sh 文件也复制过来。Morphline 在分析行内容时使用了 grok - patterns. conf 文件。该文件在/opt/examples/flume/morphlines/dictionaries/目录下，也可将其复制到 flume_conf 目录中。

对复制进来的 flume. conf 进行的主要修改如图 9-23 所示。

修改 morphline. conf 的文件如图 9-24 所示。

morphine. conf 中主要修改这几个位置，如图 9-24 方框中所示。

collection 应改为 info，因为在前面使用 solrctl 创建的集合称为 info；dictionaryFiles 改为 grok - patterns. conf 文件当前所在目录，即方框中的目录名；message 改为当前抓取信息的格式，其中

```
%{BOOK_NAME:book_name}%{URL_REL:url_rel}%{TSTAMP:id}
```

这三部分用 tab 分割，与抓取时写入 access. txt 的形式是一样的。其中，%{BOOK_NAME:

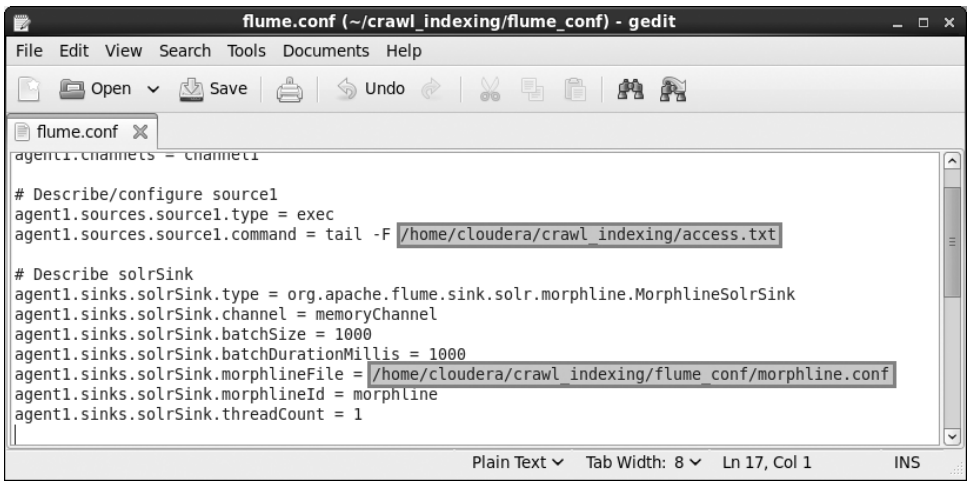


图 9-23　对 flume_conf 的修改

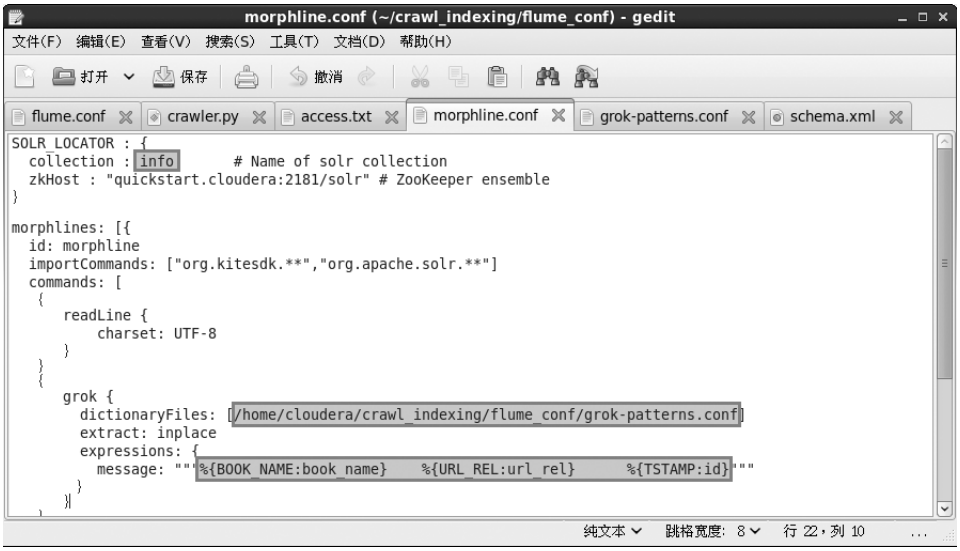


图 9-24　修改 morphline. conf 文件

book_name} 表示 access. txt 中的一行按 tab 分割后，第一个字段将会使用 grok - patterns. conf 中 BOOK_NAME 对应的正则表达式规则进行匹配，匹配结果将作为 Solr 中 info 集合中的 book_name 域，%{URL_REL:url_rel} 为处理分割后的第二个字段，%{TSTAMP:id} 为处理分割后的第三个字段，匹配的规则在 grok - patterns. conf 中 TSTAMP 对应的正则表达式中，匹配的结果将作为 Solr 中 info 集合的 ID 域。

接下来就是在 grok - patterns. conf 文件中添加 BOOK_NAME、URL_REL 和 TSTAMP 对应的正则表达式。在该文件中提供了很多类型规则供参考，本例增加的规则如图 9-25 所示。

有了上面两个配置文件就可以运行 Flume 了，为了运行时输入命令更为方便，将 Flume 启动的命令写到 flume_cmd 文件中，如图 9-26 所示。

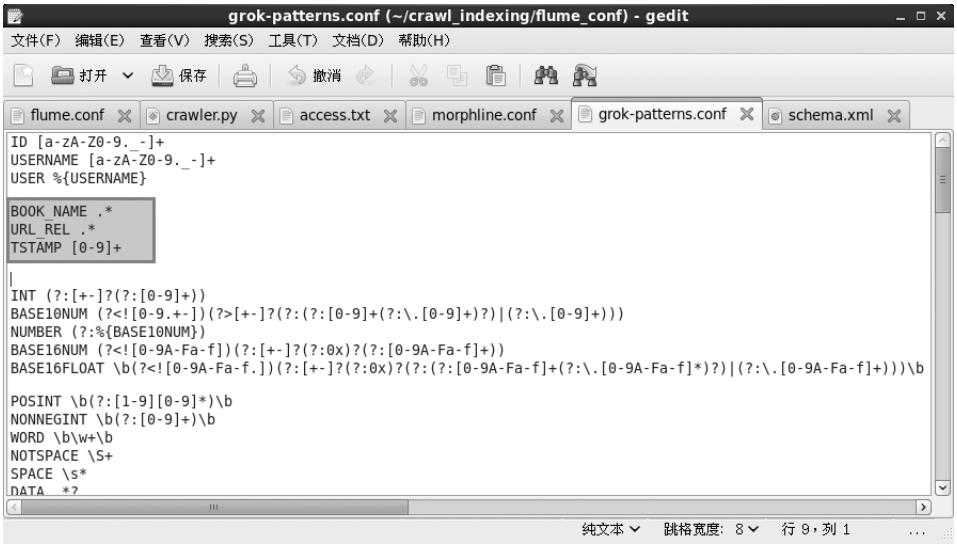


图 9-25　本例增加的规则

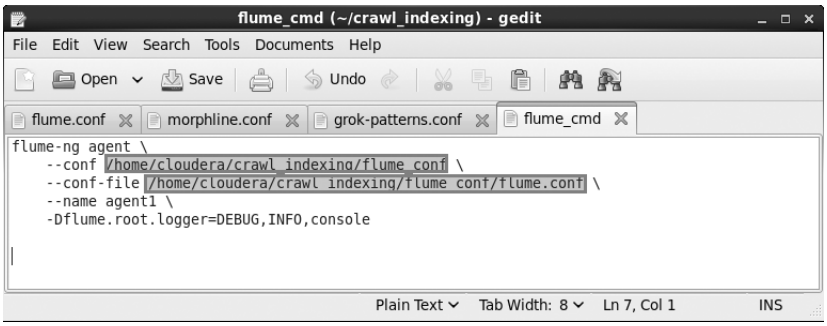


图 9-26　Flume 运行的命令

注意图中方框表示的配置文件目录和配置文件的名字，使用时可直接将上面文本文件的内容全部复制到 shell 中，如图 9-27 所示，直接回车即可。

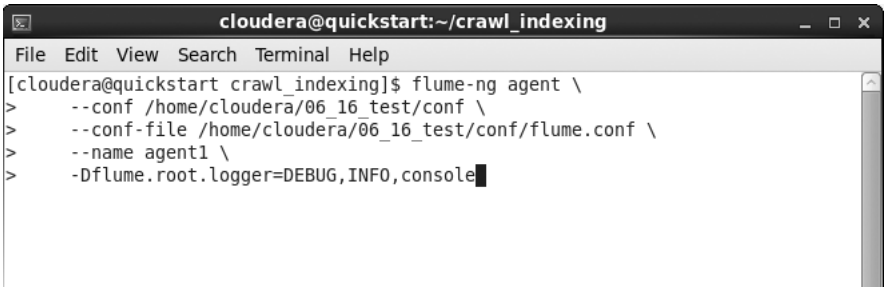


图 9-27　在 shell 中执行命令

此时，如果 crawler 没有启动，则可以重新启动。抓取一些记录后，可以刷新前面没有

显示结果的搜索页面，就可以看到结果了，如图 9-28 所示。

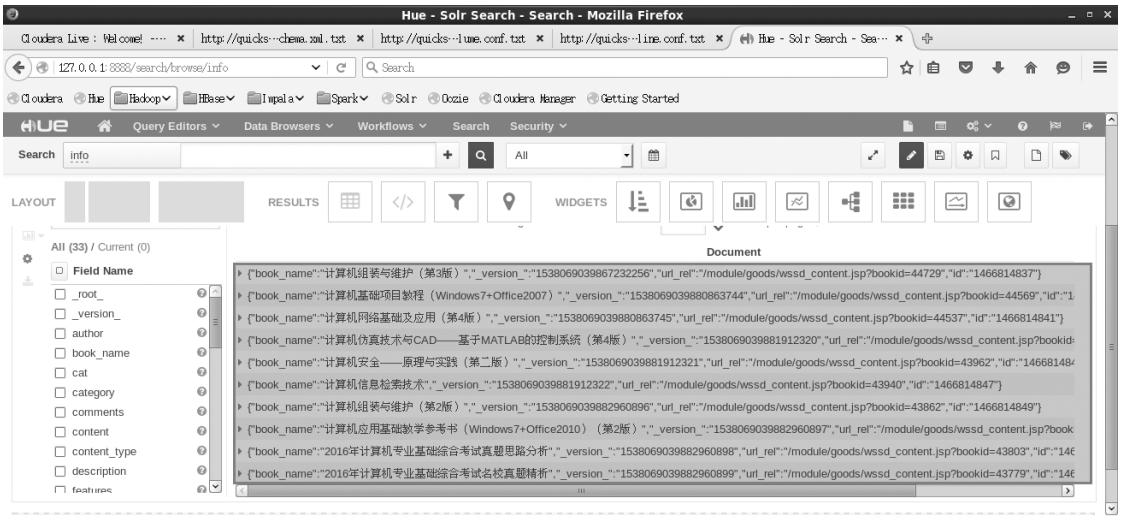


图 9-28　数据已被索引

可以在左上角的 Search 搜索框内输入“book_name:技术”，注意输入时不要输入双引号，可以检索出此时已经被索引并保存的各条书名中包含“技术”的书籍记录，如图 9-29 所示。

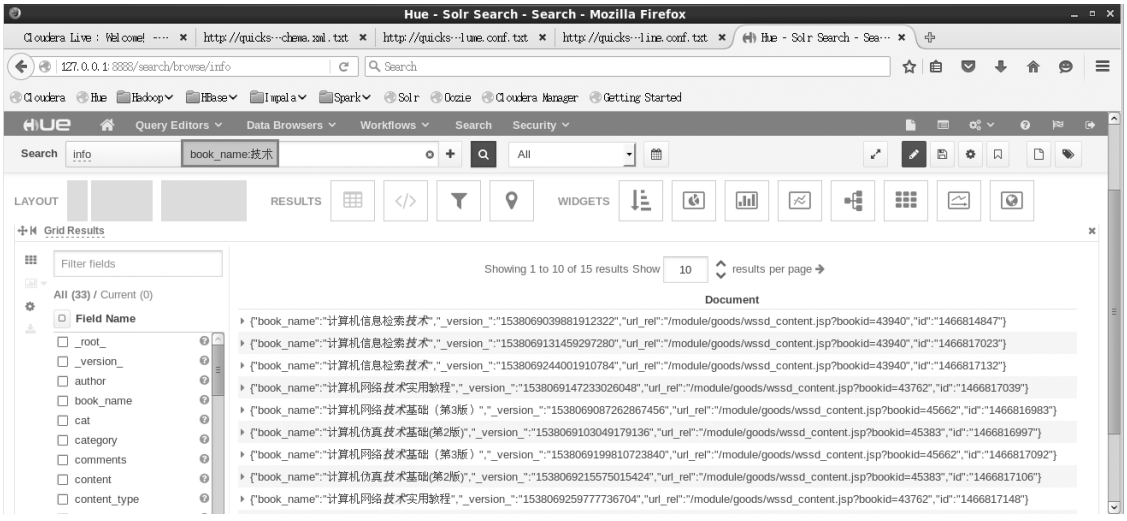


图 9-29　数据搜索结果

实战

下载和使用 Cloudera 提供的 Vmware 虚拟机环境，根据本节的内容配置一个实时抓取和索引的系统。

反侵权盗版声明